WxSmith tutorials

From CodeBlocks

Jump to: navigation, search

Welcome to wxSmith tutorials page. Here you can learn how to create GUI applications for wxWidgets platform using wxSmith. At the beginning we will start with some basic tutorials which will show what can you do with wxSmith and how you can create simple applications. I hope that there will also be some advanced tutorials for those who know wxWidgets very well.

[edit] wxSmith and wxWidgets basics

On this tutorials you will learn basic things about wxSmith and wxWidgets.

- Tutorial 1: Hello world
- Tutorial 2: Working with items
- Tutorial 3: Building more complex window
- Tutorial 4: Working with multiple resources
- Tutorial 5: Using wxPanel resources
- Tutorial 6: Accessing items in resource
- Tutodial 7: wxSmith tutorial: Creating items with custom paint and mouse handling

Category: WxSmith Documentation

Views

- Article
- Discussion
- Edit
- History

Personal tools

- Log in / create account

WxSmith tutorial: Hello world

From CodeBlocks

Jump to: navigation, search

Contents

[edit] "Hello world" Tutorial

Hi, in this tutorial we will write very basic application that will use wxSmith(which is a smart tool to support GUI design), called "Hello world". Since this term reflects to the very first step for some coding language, it may also be treated as your first step to learn wxSmith.

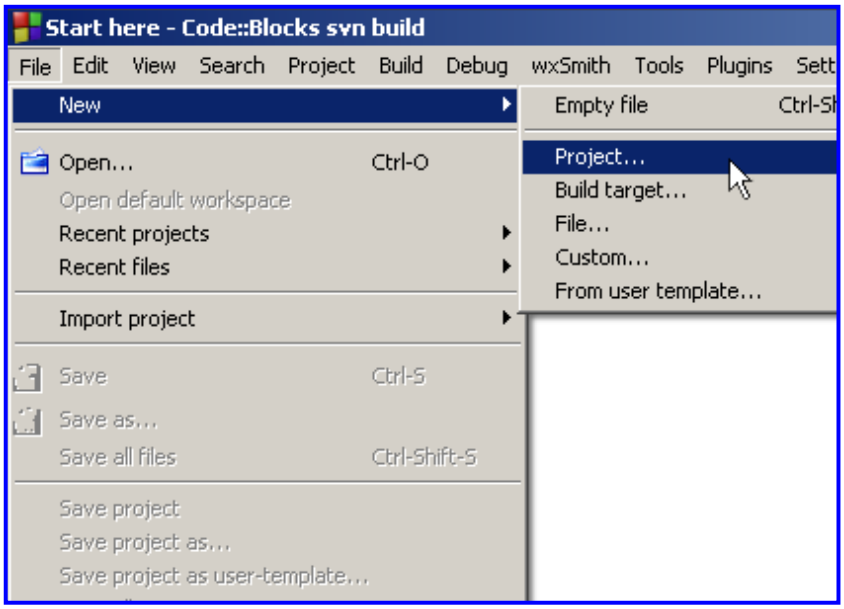
Before we start you will have to compile wxWidgets (or download precompiled binaries and header files). It's described very well on the wiki pages below:

- windows user see the wiki page:Compiling wxWidgets 2.6.2 to develop Code::Blocks (MSW) or Installing Code::Blocks from source on Windowsor WxWindowsQuickRef. There are two method to install the wxWidgets libraries, one is you can download the wxWidgets source and compile your self. The other is you can download the wxPack from [1], then install it, the wxPack will give a pre-compiled wxWidgets library, so you can save a lot of time.

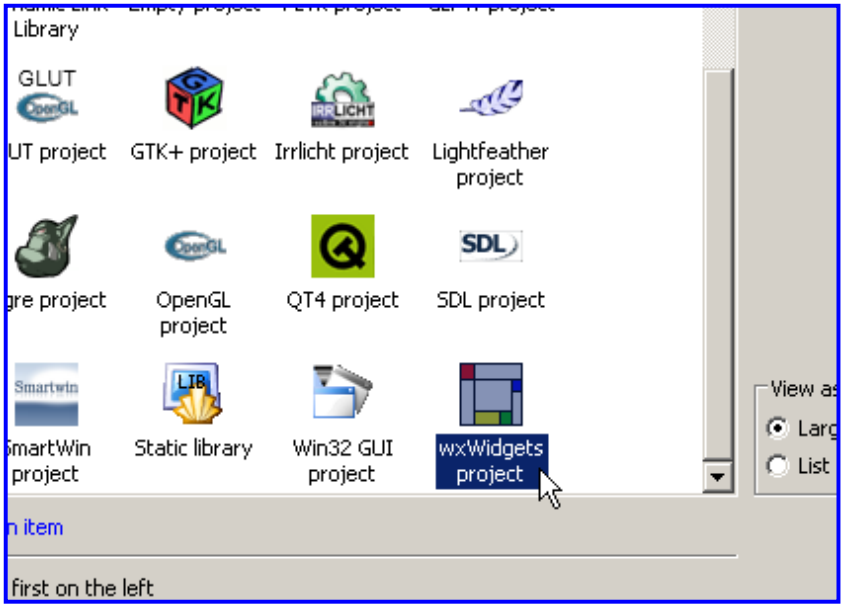
The most important thing is that you have get the wxWidgets ready working with code::blocks. Now, let's make some project :)

[edit] Creating wxWidgets project

The easiest and the preferred way is to create wxWidgets project using project wizard. It stars like creating any other project using *New project* menu:

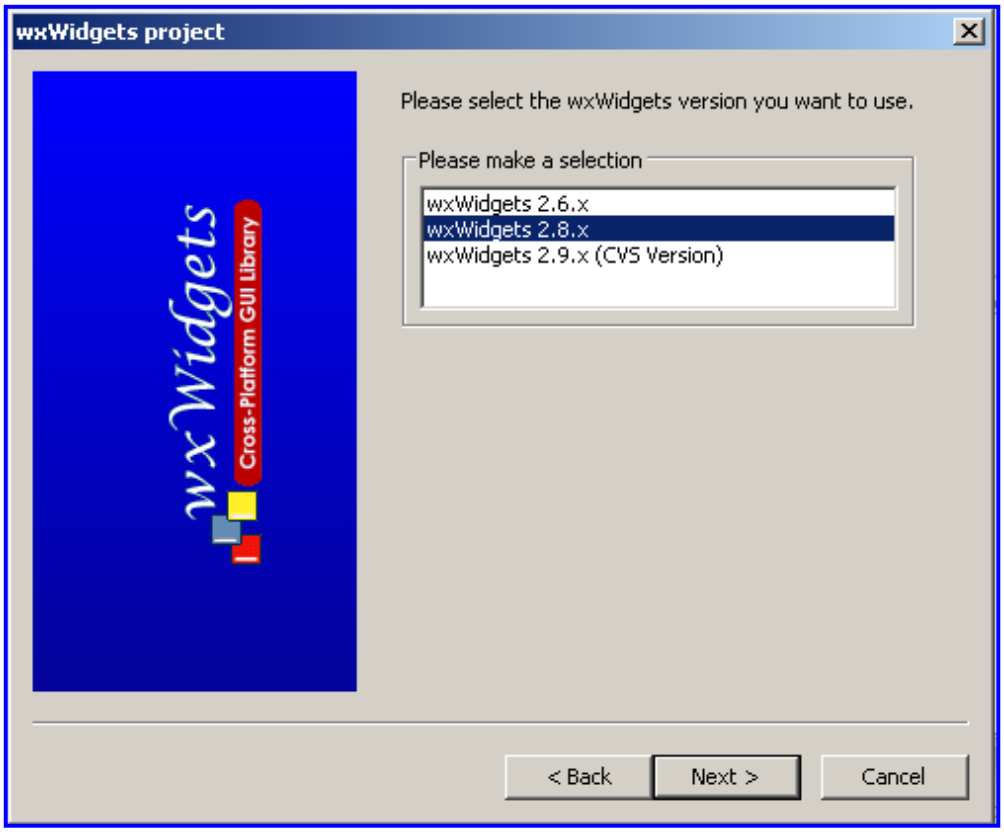


Then to create wxWidgets application, let's choose wxWidgets icon:



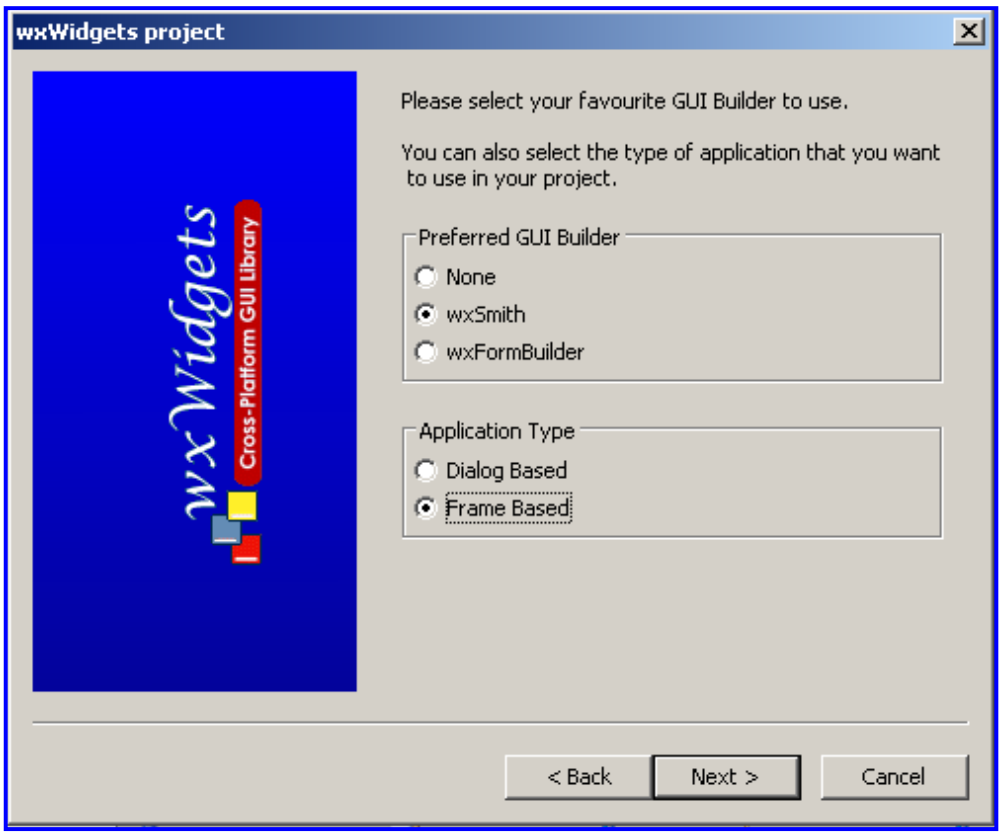
After you click "Go" button, Code::Blocks will bring up the wizard window. It consist of several pages where you can enter some options for your project. I'll describe few pages better since they may cause problems for new users.

At the very beginning the wizard asks you which version of wxWidgets you want to use:



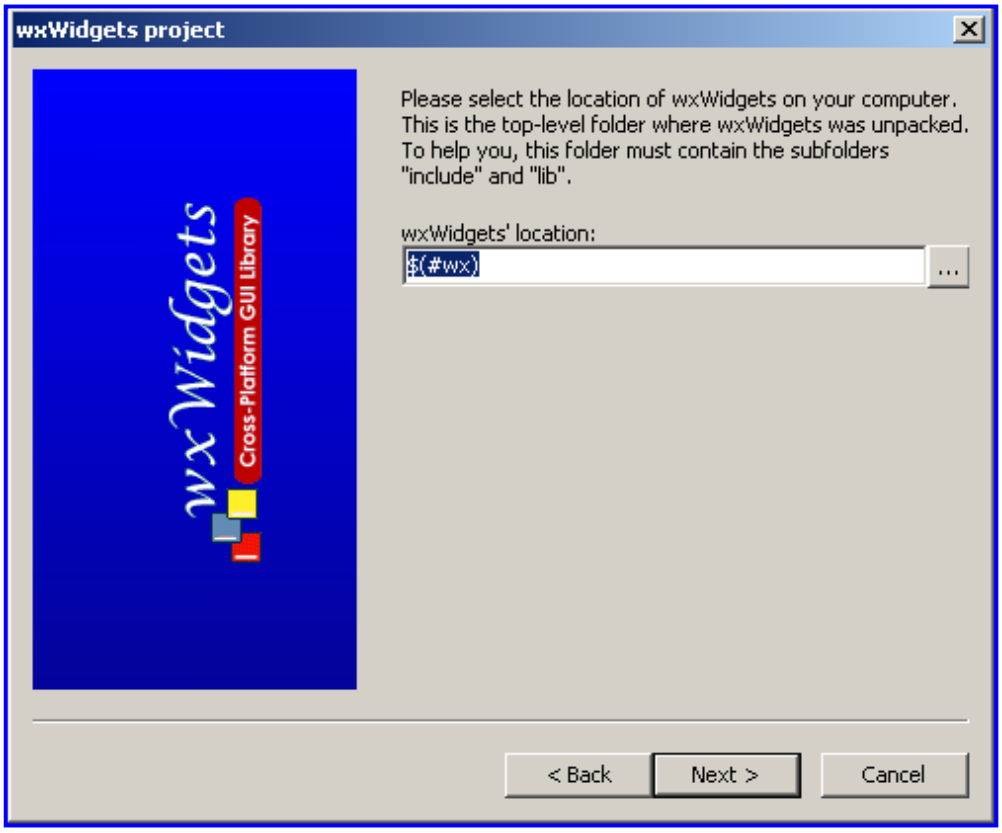
The best option for now is to use wxWidgets 2.8.x since it is stable and has few features not present in wxWidgets 2.6.x series. Also wxSmith aims to provide best support for latest stable version of wxWidgets.

On another wizard page you choose what GUI designer will be used and what type of application should be generated:



If you want your application to work with wxSmith you should check the wxSmith designer. You can also use powerful wxFormBuilder, but this program is not covered in tutorials presented here. Another option let you choose if your application should be created using wxFrame or wxDialog. Both options will lead to valid wxWidgets application but wxFrame should be preferred since it's more natural for wxWidgets and wxDialog-based applications are not as flexible as wxFrame-based ones.

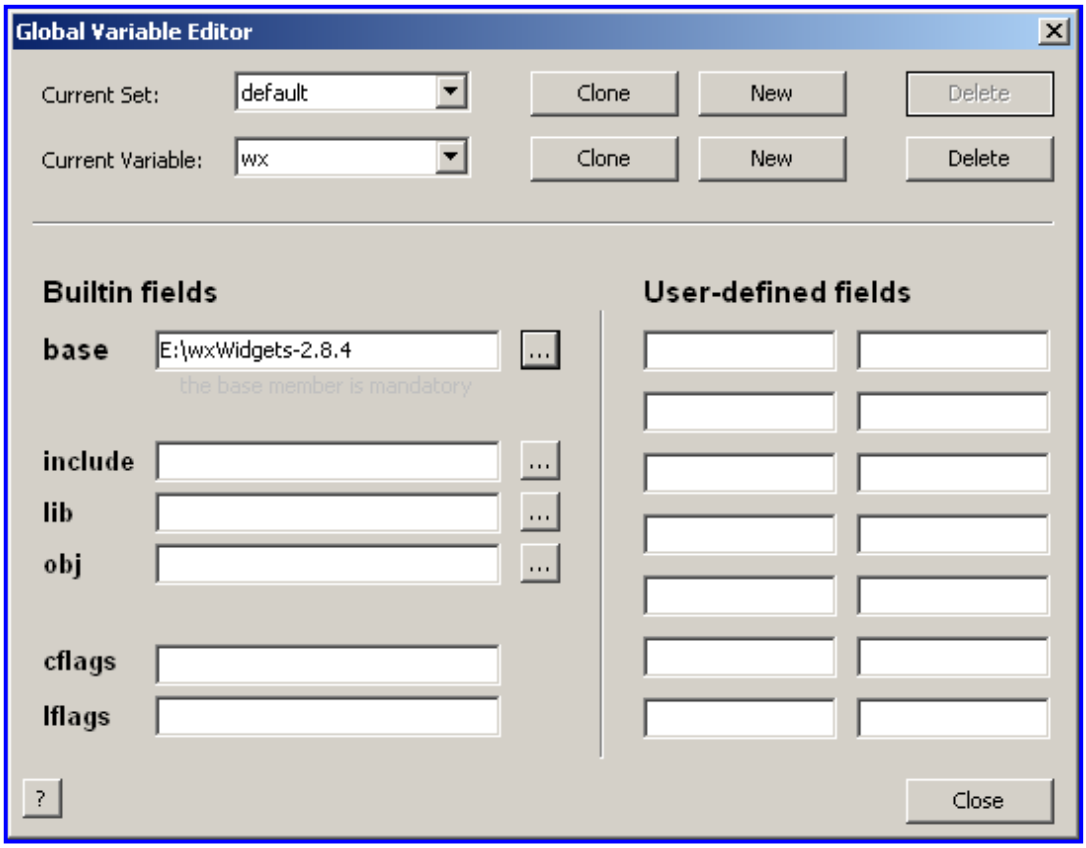
On another page you may choose the location of your wxWidgets library:



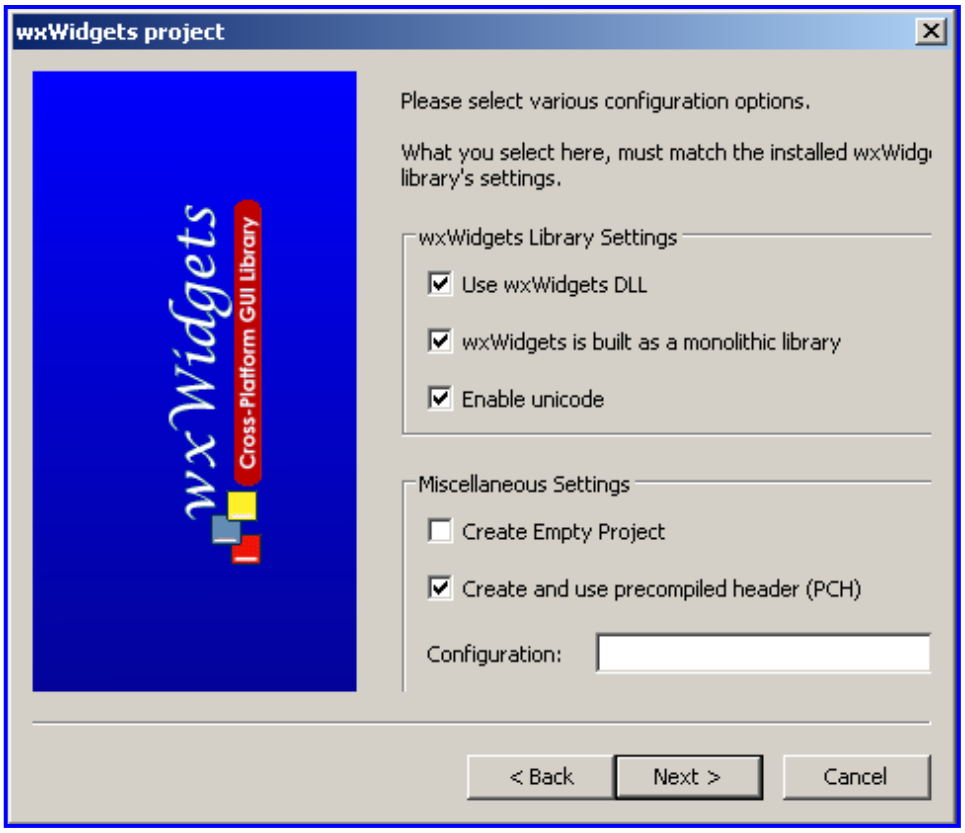
You can choose real directory here, but that may lead to problems when your project should also compile on other computers. Another choice is to use **Global Variable system** (to open the dialog below or modify or change this global variable, you should click on the menu **Settings->Global variables**) available in Code::Blocks.

When you use it, the location of wxWidgets library is read from such variable. This mean that wxWidgets directory is not hard-coded inside your project, but is stored inside global configuration of Code::Blocks. When your project will be opened on other computer, it will use it's configuration. The easiest option to use global variable is to leave *$(#wx)* what means to use global variable named *wx*. This name is reserved for wxWidgets and should be chosen in most cases.

When you use global variable for the first time, it will ask for wxWidgets directory. The only thing you need to do here is to enter valid location in *base* field. This is required only for the first time since Code::Blocks will automatically use this setting for other projects you create.



On the next step, wizard will ask you for configuration of wxWidgets library. Since wxWidgets may be compiled differently, each compilation type should also be threated differently inside your project. This step may be little bit confusing for new users since it require some knowledge about build process of wxWidgets. But if you used wiki pages that I've put at the beginning of this tutorial to compile wxWidgets, set your configuration to this one:

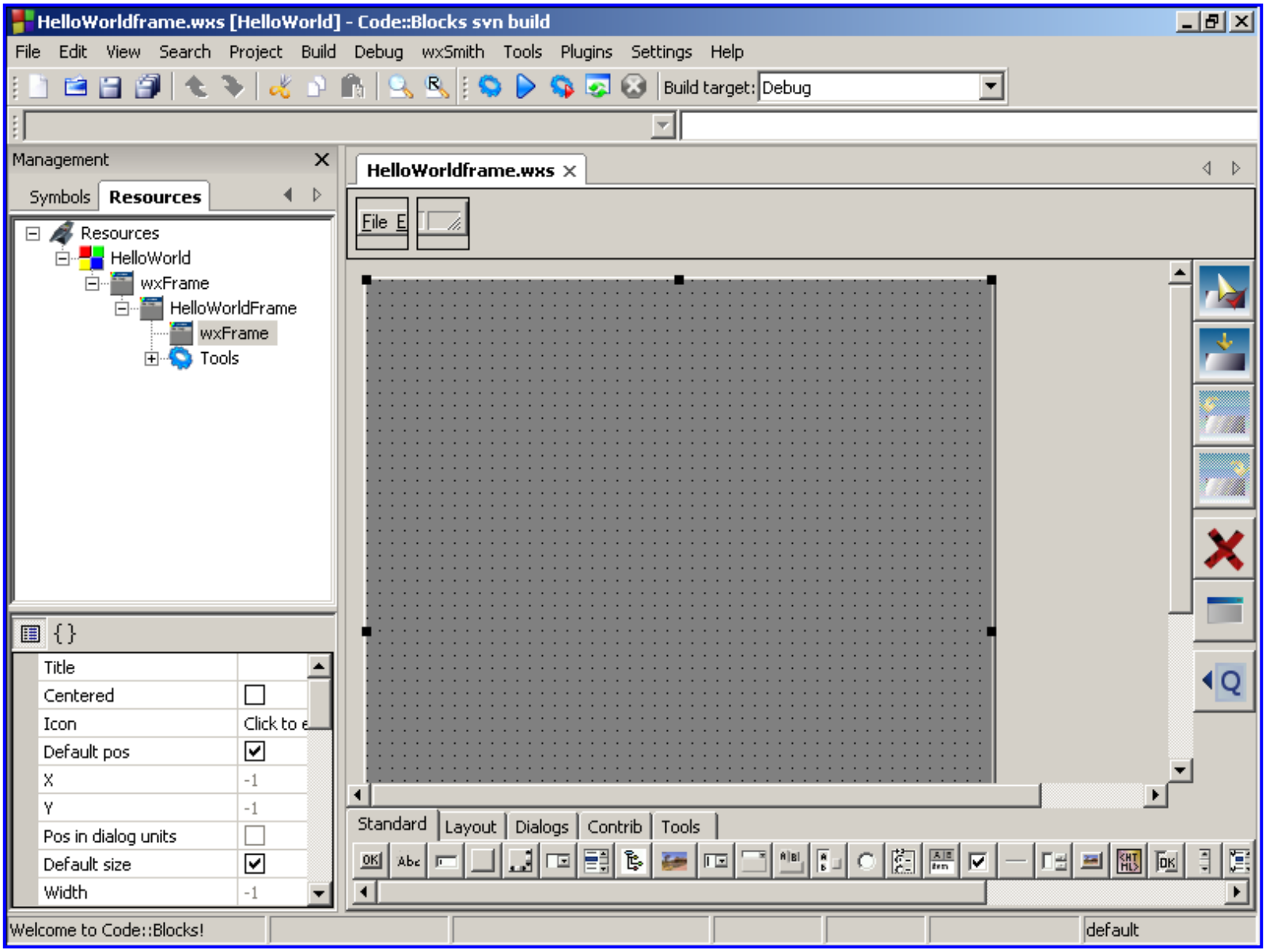


Here is a brief explanation of these options

- **use wxWidgets DLL** : If checked, when you create an executable file, it link to wxWidgets dynamic link library, So, your exe file should run with some wxWidgets DLL files. If unchecked, your executable file's size will be larger, but it can run without wxWidgets DLL's support.
- **wxWidgets is built as monolithic library** : If checked, your target build file will link to the monolithic library of wxWidgets. see WxWindowsQuickRef#wxWidgets_Build_Options_Explained for more details.
- **Enable unicode** : If your program will support some international language other than English, such Chinese, Japanese, you can check on this option.
- **Create and use precompiled header(PCH)** : It will speed up building your program. see PCH for more details.**Note** that I've also checked the option to use precompiled headers (PCH) since it can really speed-up the compilation of wxWidgets applications on some compilers (especially GCC).

When you follow another wizard pages, you may occasionally see message box saying that debug configuration can not be found. This is because the wxWidgets library was build using release mode (the *BUILD=release* option used for make). If you want real debug library, you can recompile your wxWidgets by using *BUILD=debug*. Project may also be tuned to work with release version on both debug and release build targets (the way to do this is presented at the end of this tutorial).

After project is created and initialized properly, Code::Blocks will look like this:



Now when you select *Release* build target and press Compile, it should create wxWidgets application. But before you run it, you will probably need to copy wxWidgets dlls (from *lib\gcc_dll* inside directory where your copy of wxWidgets resides) to project's directory. When your application is ready and runs without problems, we can jump to the next point.

[edit] Let's add some fireworks

Now it's time to fill the empty frame with some "Hello World" message. This text could be added directly into frame, just like in most commercial GUI designers. But we will learn to use sizers. But what's that?

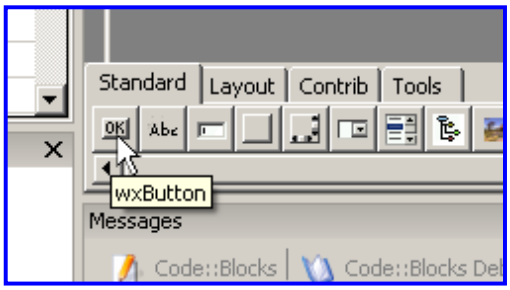
If you have been working with java you will remember something called Layout managers. Implementation in wxWidgets differs a little bit but it does almost the same.

Ok, but let's put some explanation here: Usually when adding items into windows you must specify the item's position and size. wxWidgets tries to automate this process and it uses sizers for that. They are automatically positioning and sizing window items and they can also set the size of window to fit all items inside. Sizers have one big advantage. When you write cross-platform applications, you cannot assume that fonts, buttons, etc. are sized the same on different operating systems. This can even occur on the same platform when using window themes. When you use sizers, you don't have to worry about that. All sizing is done automatically. And one more thing - sizers can even reposition and resize window items when the main window changes size.

Now since we have some background knowledge, it's time to add something.

When you look at the wxSmith editor, you'll notice that the area of window is surrounded by eight black boxes.

These black boxes are surrounding a currently selected item. In out case it's a whole window. Adding new item is simply done by clicking on one of the buttons in the palette at the bottom of the editor and then clicking some position on the resource.



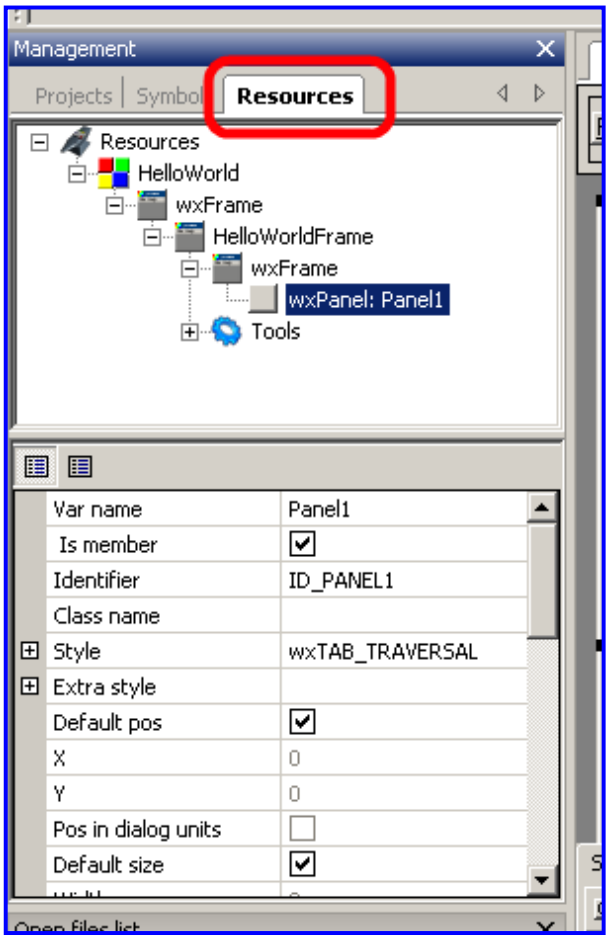
As you can see here, when you keep your mouse cursor over a component, it will show it's name. That may help to find items when you're not familliar with their icons.

Let's add new wxPanel component to our window - this is fourth icon from the left. You may see that when you click the wxPanel item and then move cursor over the resource, some part of the editor will change colour. That's intentional and used to show onto which part of resource new item will be added.

After wxPanel is ready you will notice that the background colour of window changed to light gray. Now our window will look much more like other windows. Note that currently the selection changed into added panel but since it's size is equal to frame, it's hard to find out what's

really selected. To make sure that wxPanel is selected, click somewhere on light gray area of our window (Note to Linux users: wxFrame and wxPanel usually have same background so it may look like nothing has changed, you may ensure that there's wxPanel by looking into resource browser).

If you want to see the selection exactly or select items that overlap, you can use resource browser. To show it, click on resources tab located in same place where project browser is:



Resource tab consists of two parts. First is resource tree which shows structure of resource and second one is property browser which is common in GUI designers. Inside resource tree you can find currently selected item. In our case it is wxPanel class we added before. Selected item also shows the name of this item: Panel1. This name is basically the variable which can be used to access this component in your code.

We added new item by selecting the place for new item. This is the default wxSmith's behavior. However, there is other mode of adding items. In this mode, new item is added right after you click on palette's button. The position of new item is relative to current selection - just like cursor in text editors. Basically we can add new item into three different positions relative to selection:

- Before the selected widget
- After the selected widget
- Into the selected widget.

This can be changed by clicking one of four "placement" buttons on the right side of editor:

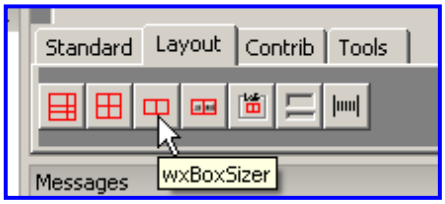
The upper most is the mode we used - "point by mouse", next one means "add into", following is "add before", and the last one means "add after". Current mode is spotted with red check mark on the button. Sometimes few selection modes are disabled. This means that they are not valid for currently selected item. For example you can not add any item into wxButton component since wxWidgets declares it as the one that can not have child items. And when the main component (the window) is selected, you won't be able to add anything after it or before it.

Now let's assume that we've made a mistake and want to delete item that we have just added. In fact we have a small mistake and will have to delete wxPanel. To this, click on the red X button:

When you click this button, it deletes current selection, so if the panel didn't disappear, make sure it's selected and click the button again. Now we should return to the state at the beginning.

The mistake made here is that we've added wxPanel directly into main frame. But as I've said before, we will be using sizers to manage window's content. To use sizers, first we have to add sizer into main frame and then add wxPanel into it.

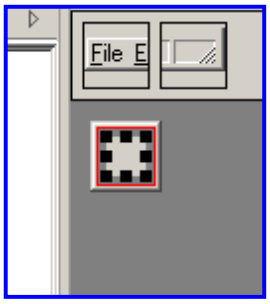
Sizers are available in the **Layout** tab on the palette. Switch to it and select wxBoxSizer:



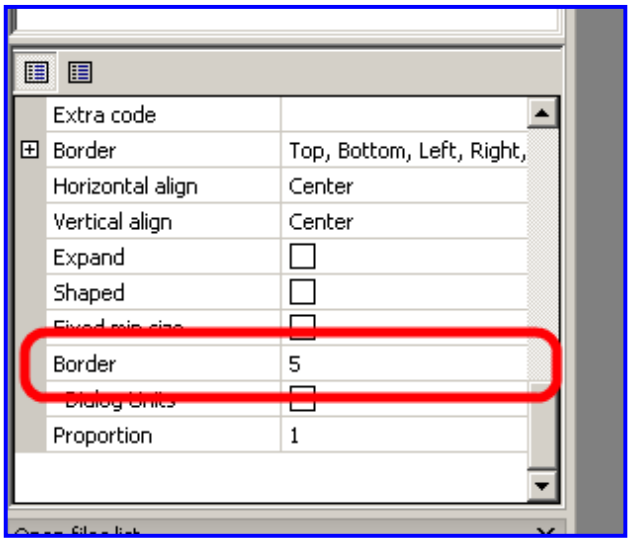
This sizer tries to position items in one line, either horizontal or vertical. Horizontal is by default and we will use this.

After adding sizer, two things have changed. First is that our window has now red border. This mean that there's sizer attached to it. When you look into resource browser you will also see that sizer has been added (you may need to click on the editor area to refresh the selection on resource browser). Second thing is that size of window has shrinked to small box. That's correct since sizer is also responsible for resizing items.

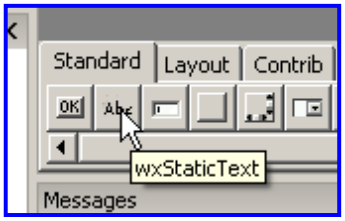
Now let's add wxPanel. Make sure that we will add it into sizer, not main frame. After we do this, you will see something like this:



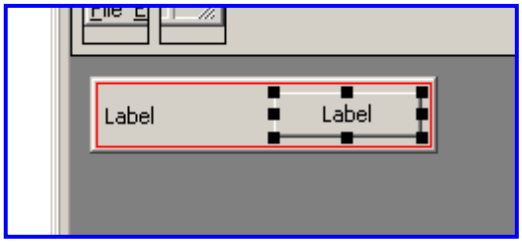
Here we can see our panel surrounded by drag boxes. There's also some dark gray border around it (it won't be visible on Linux). Each item added into sizer can be surrounded by such border. It's useful when you want to have spacing between items. But in our case we should get rid of it to get more proper window. To do so, we will have to change property inside property browser. Search the border size and change it to 0:



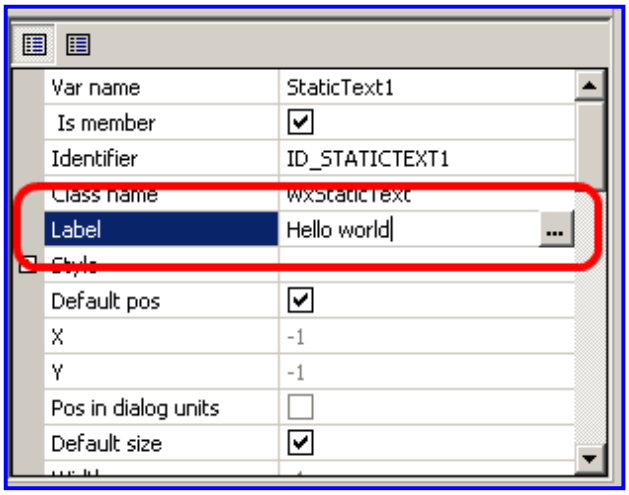
Now when we have our panel adjusted, we can finally add "Hello world" text. Since we will also use sizers to manage items added into wxPanel, we have to repeat addition of wxBoxSizer into wxPanel. After sizer is on it's place, switch back into **Standard** tab on the palette and add wxStaticText:



Because this text looks so *lonely*, let's add button next to it by clicking on wxButton. Note that if you still have "point by mouse" insertion mode turned on, half of wxStaticText item will be coloured in light blue when you put mouse over it. This is a hint for you: when the left half of text is highlighted, new item will be added before it, and when the right half is highlighted, it will be added after the text. So let's add button after the text. Now we should have this content:



Now let's finally set our "Hello world" text. wxStaticText control which we added before has "Label" text on it. It is default text set to all newly created wxStaticText classes. We will change it using property browser similarly to changing border size in wxPanel class. To do it select our text first, find **Label** property and change it to "Hello world":

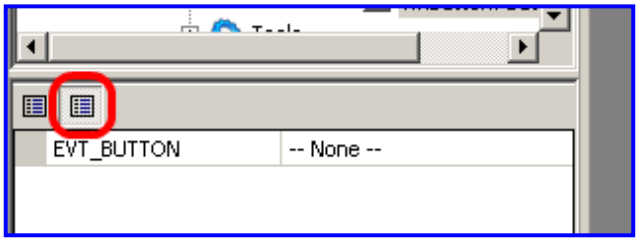


You can also set multi line text by clicking on "..." button or putting "\n" inside text (just like in c strings).

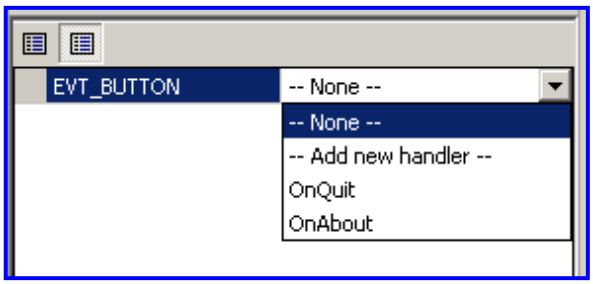
[edit] Let's react - making close button

Now let's do something with the button. Since buttons usually react when somebody clicks them, let's make it close the window. First thing we should do is to change text shown on button to notify user about function assigned to it. It is done just like in case of text - select button, find "Label" property and set it to "Close" (or anything you want ;)).

Second thing is to assign event handler to this button. Basically event handlers are function which are called when some "event" occurs. In our case instead of "event" we could say "click on button" because that we want to process. As many GUI designer, wxSmith can automatically generate such handlers. They are accessed through property browser just like other properties. To switch to events you have to click second icon at the top of property browser (don't worry, those icons are going to change to something more self-explanatory):



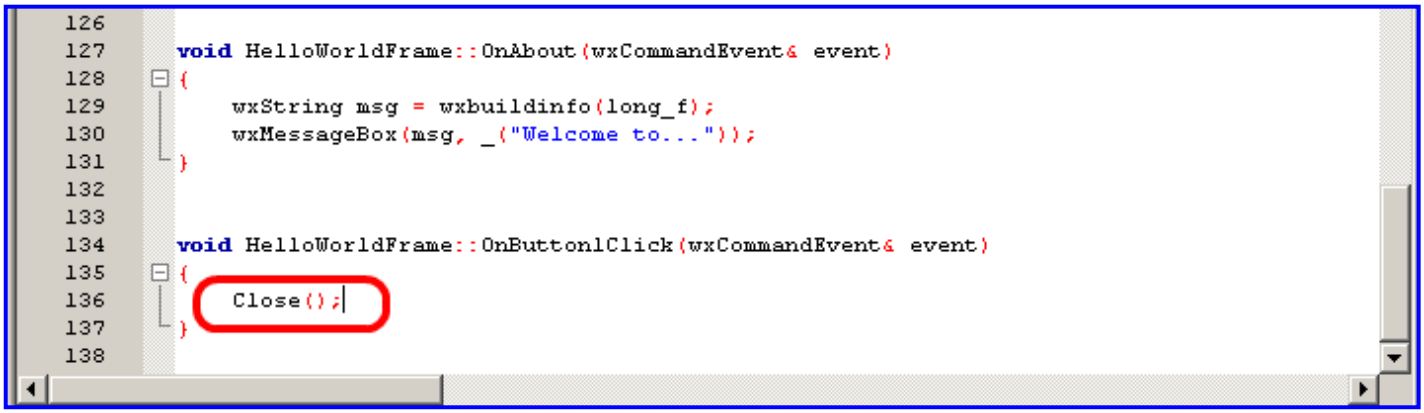
After you click on this icon, you will see all events supported by currently selected item. wxButton class support only EVT_BUTTON event so only this one is shown. When you click on this event and expand combo box you will see available options and handlers for this event:



When you select **-- None --** there won't be handler function automatically assigned to this event (it may always be set manually), selecting **-- Add new handler --** will generate new handler function. Below are showed other event handlers known to wxSmith which can be assigned to this event. We can see here OnQuit and OnAbout. Those handlers were created by wizard. So let's create new handler. We could use one of

those defined, but since we haven't looked into source code yet, we don't know what they do. After you select new handler option, you will be asked for name of handler. This is simply name of function which will be called as a reaction for event.

When you enter new handler name, wxSmith will generate new handler function, open file with proper source code and put cursor inside handler function (note that you may need to press up or down cursor to show the line with cursor). Note that this function is member of class called for example **HelloWorldFrame**. wxSmith creates one such class for each resource. Since event handler function is member of this class, you have access to all the window from it. So when we want to close our window we just call **Close** function:



```
126
127  void HelloWorldFrame::OnAbout(wxCommandEvent& event)
128  {
129      wxString msg = wxbuildinfo(long_f);
130      wxMessageBox(msg, _("Welcome to..."));
131  }
132
133
134  void HelloWorldFrame::OnButton1Click(wxCommandEvent& event)
135  {
136      Close();
137  }
138
```

After compiling our project it will show our "Hello world" and "Close" button.

This is end of this tutorial, I hope that it showed some basics of wxSmith. If you have any problems with it or think that some parts are unclear, please contact me (byo) on forum.

Retrieved from "http://wiki.codeblocks.org/index.php?title=WxSmith_tutorial:_Hello_world"

Category: WxSmith Documentation

Wxsmith tutorial: Working with items

From CodeBlocks

Jump to: navigation, search

Contents

[edit] Working with items

Hi. In the previous tutorial we learned how to create a small application using wxWidgets. We created a window with a few items inside. In this tutorial we will focus on wxWidgets items which form a resource. I'll show how you can add items into a resource, how to change them, and finally I'll describe a few of the most important and most useful items.

Note that all screenshots presented here were made on Linux but they should not be very different from the windows ones.

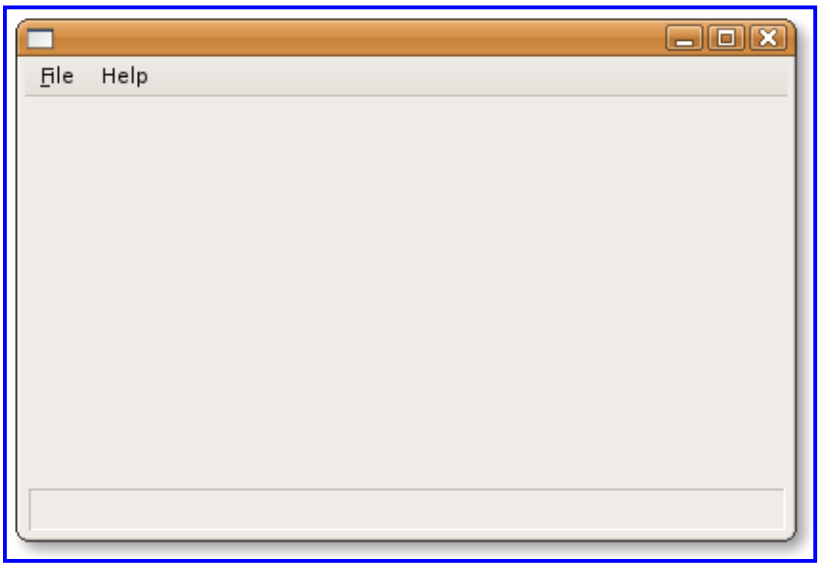
Ok, let's start.

[edit] Create new application

We will start with empty wxWidgets project as our blackboard. Steps required to do this were described in previous tutorial so if you have any problems you can look into it.

Let's select File->New->Project menu, and choose wxWidgets application there. When you're in the wizard, make sure that **Preffered GUI Builder** is set to **wxSmith** and **Application type** is set to **Frame Based**.

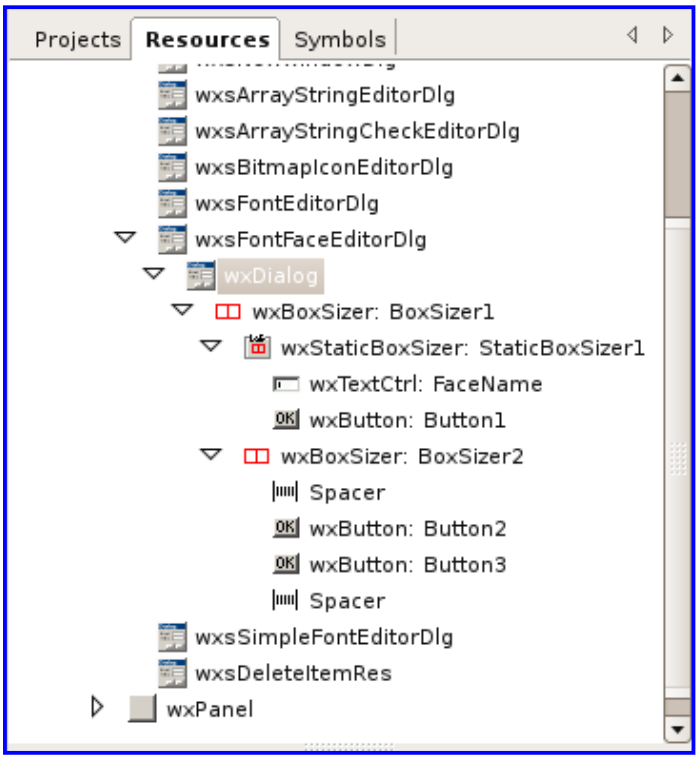
After the wizard generates a new project you should be able to compile and run it. The application should look like this:



[edit] Working with items

Each resource in wxSmith is made of items. All your buttons, text boxes, lists etc are items. Also some more abstract things like sizers, spacers (used to add empty space), menu entries or even timers are also called items in wxSmith. So basically anything that's inside the resource may be called an item. I'll also use the term component since it reflects the valid meaning too.

Actually each resource has one root item: for dialog window it would be wxDialog item, for frame window it would be wxFrame item and simillarily for panel it would be wxPanel item. If you walked through previous tutorial you probably noticed that some items may have children - for example you insert sub-menus into menus, some items into sizers and so on. Those items may also have their children. This forms a structure of tree which is directly represented inside resource browser:

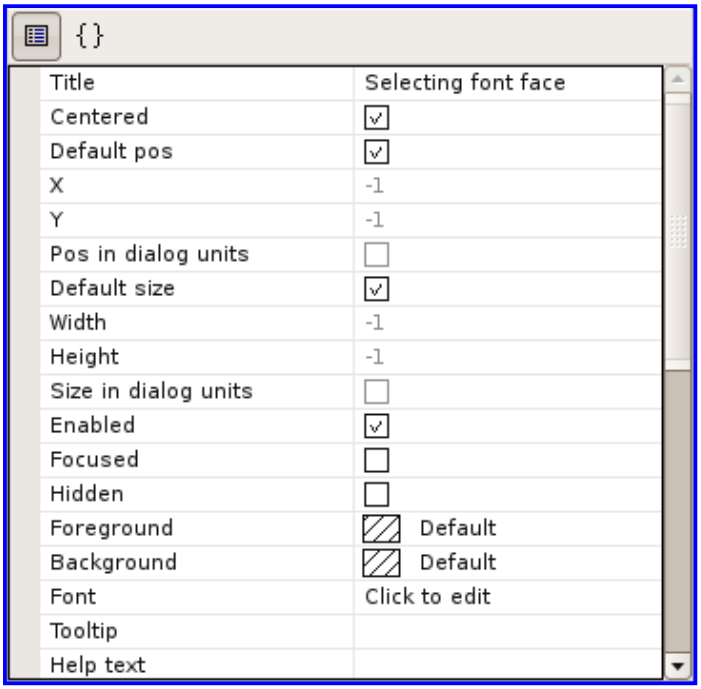


In the picture above, you will see an example of such a tree. wxFontFaceEditorDlg is the currently opened resource with a root node of type wxDialog. We can see that this resource consists of few sizers, a few buttons, one text area and some spacers. It's definitely a very simple resource. For more complicated resources, there could be more than 100 items and the tree browser may be helpful when locating them inside the editor.

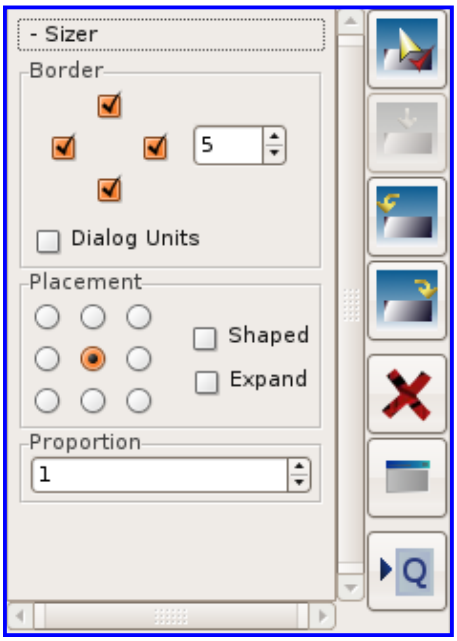
Each item has its properties - these are usually some values which describe the item. Some of them may affect the way item looks, some may change

the item's behavior, and some may be dedicated to programming facilities. You will also find that there is a set of properties available for almost every item - that's correct since many properties are generic like position and size.

wxSmith allow you to edit items inside the properties browser (the window which is usually under the resource browser). If you select an item either by selecting it in the resource browser or by clicking on it inside editor, the properties editor switches to edit its properties. Here's an example of the property browser:



You can also use the Quick-Properties panel available after pressing the "Q" button on the right side of the editor:

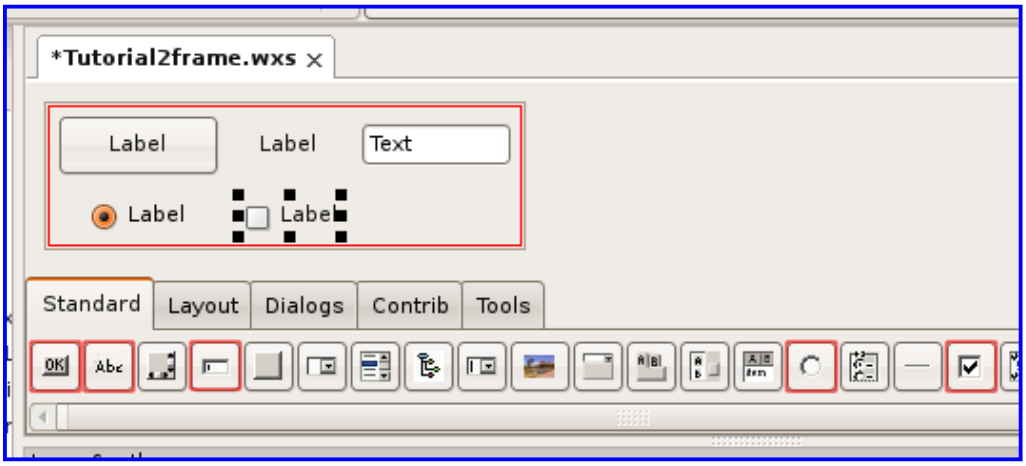


The purpose of this panel is to provide most common properties in user-friendly way. Unfortunately it's rarely used in wxSmith now and you will probably find it usefull only while adjusting sizer-related settings.

The last and the easiest one is to operate directly on item inside editor - you can grab drag boxes and change item's size. You can also move item into another position by simply dragging the item into another place.

[edit] Excercise: playing with items

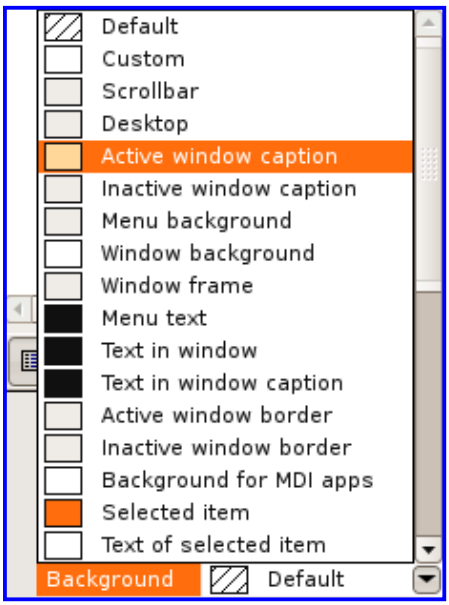
Ok, let's do some excercise. We started with empty window. First let's add wxFlexGridSizer into it - it's located in Layout pallete (you can find informations on how to add items in previous tutorial). Now let's add following items into it adding each one after the previous: wxButton, wxStaticText, wxRadioButton and wxCheckBox. The result should look like this:



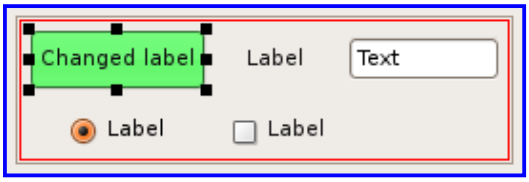
I've selected added items with light red border on the palette - all can be found on Standard category.

Now as we have our playground, let's change some items. First thing we will do is to change label and colour of the button. To do this let's click on it and move to properties window.

Label is the first property and it's quite easy to change - you can do it by clicking on the "Label" on the right and just typing new text. Colour is little bit harder to find. Usually items have two colours - Background and Foreground colour. Background, as the name says, is used to fill item's background, Foreground on the other hand is used to paint some content onto background. Note that there's no strict interpretation of these colours so you should experiment to find out their meaning. In button, Background is what we need. So let's find Background property - it's value can be changed by using combobox:



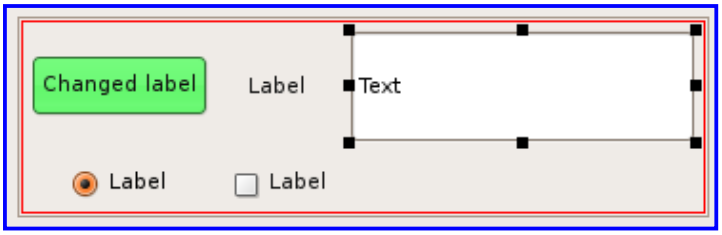
On the list you may find set of predefined system colours. When you choose one of them, they will be read from current system settings making you application compatible with system theme. There are also two special entries - Default (which means that we do not want to set our colour) and Custom. When you select Custom, wxSmith will open colour dialog where you can choose any colour you want. I have choosen light green and here's the result:



There are also other properties we could talk about, but the best way to learn them is to play with them, so I leave the discoveries to you :)

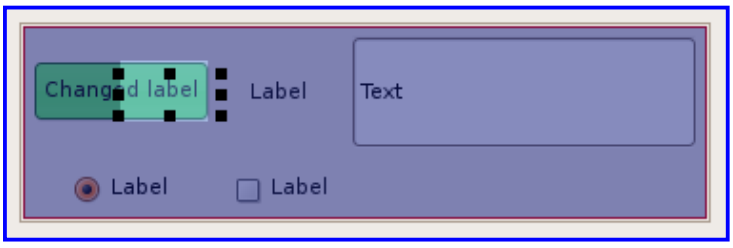
Now let's use a mouse.

Selected item have eight black boxes around it - these boxes can be used to change size of item. You will easily notice when mouse is over the black box because the cursor changes. So let's resize the wxTextCtrl (edit box):

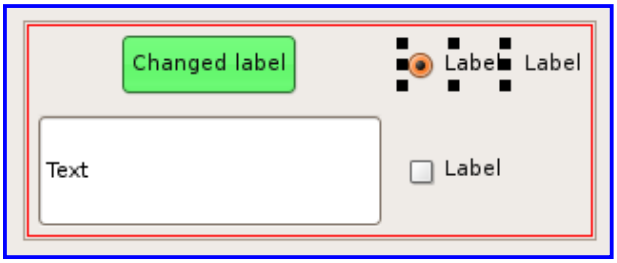


You may see that resizing item will also change properties in browser - **Default Size** is unchecked now and **Width** and **Height** is set to required values. There's also other property which seems to be untouched - **Size in Dialog Units**, which is also related to size. I'll explain it better: Ususally when you must specify size (or other size-related thing) you provide value in pixels. Alternatively you can specify values in unit called **Dialog Unit'.** *Dialog Unit* is usually little bit bigger than one pixel and it's size depends of size of current font (actually it's size of font of some item's parent which does have a font). This is usefull when you want some values to be proportional to size of text presented on window. If you check '*Size in Dialog Units*, wxSmith will take care of it and sizes will be calculated in Dialog Units instead of pixels (although you will still be able to resize items naturally).

Now let's change position of radio box. Ok, but I've said that sizer is responsible for the positioning - that's right. We have limited possibilities to change position - inside one sizer we can only reorder items, for more complex resources we can also move item between two sizers and other areas where sizers do not apply. Ok, back to our example, let's move the radio box to position where it would be right after the button. To do this click on the radio button and drag it onto button. When you start dragging you will notice that the area of sizer changed to blue colour - this is a helper to show you what will be the new parent of dragged item. When you move cursor onto button, half of it will become light-blue. This helper shows where new item will be placed - when the left half of pointed item is highlighted, dragged item will be added before it. When the right half is hightlighted, dragged item will be added after it. And when there's no highlighted item, new item will be added as the last child of the parent:



When you drop the item, it will change it's position. Note that all items after new position also moved - that's correct because default settings of wxFlexGridSizer allow only 3 items in one row (this can be changed in property browser) so items just adopted to new order:



Now let's learn a little bit about items available in wxSmith. I'll describe all supported sizers and few important components from Standard palette. Let's start with sizers.

[edit] Few words about available sizers

Currently we know that sizers are responsible for automatically setting position and size of some components. But how do they do that? Well it depends on which sizer has been used, what are the rules? The genric rule is that sizer tries to use all it's available space and place managed components in such way that they do not overlap. If the area available to sizer is to small, it requests bigger area so you can be sure that all components won't get out of the window (unfortunately you can not be sure that after such adjustments the window will be small enough to fit on the screen so be careful with that). Another rule which should be mentioned here is that sizers automatically set-up the minimum size required by the window. Basically when you edit window using sizers, you should be aware that you edit the smallest layout and without some tricks, you won't be able to resize window to something smaller.

Here's list of all sizers available in wxSmith:

[edit] wxBoxSizer

wxBoxSizer is the most basic sizer available in wxWidgets. It's purpose is to align components in one line - one next to another - either horizontally or vertically. This sizer also tries to keep some proortions between components - for horizontal sizer you can set factors which will keep proportionality of widths and the height of the sizer area will be equal to height of the tallest component. Analogically, vertical sizers keep heights proportional and the width will be taken the widest component.

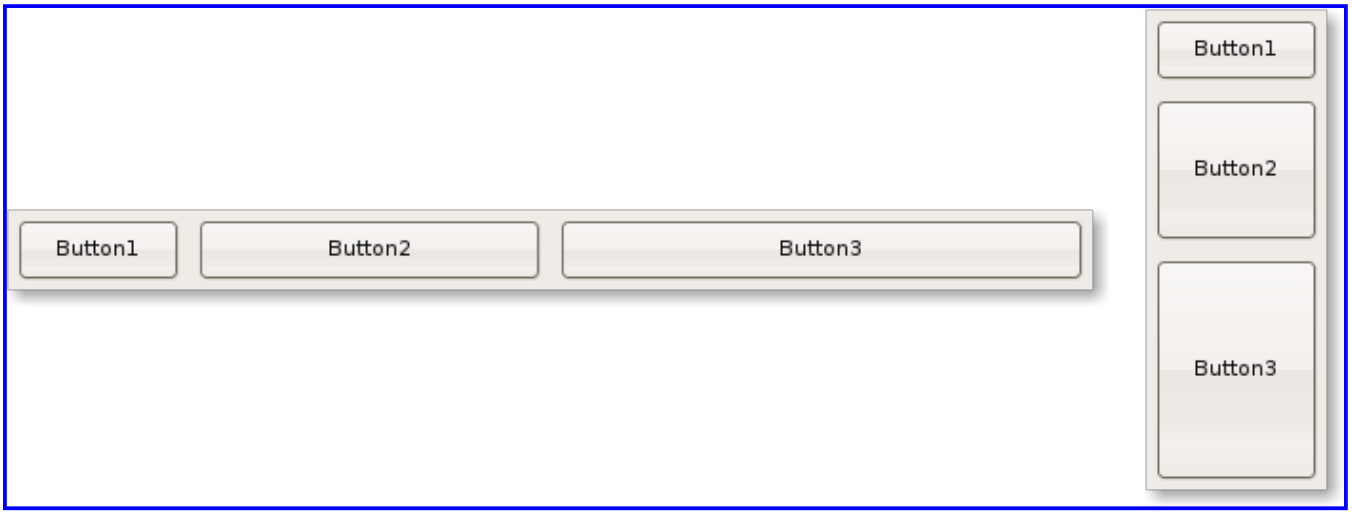
Here are examples of layout using box sizers (the first one is the horizontal one, the second is vertical one):

In that example you may see that all components have same width and height.

But let's assume that we want the Button2 to be 2 times wider that Button1 and Button3 to be 3 times wider than Button1. This can be easily done by using the Proportion property of managed components. By default the value of all components is set to 1 so they have equal size. Changing the Proportion of second button to 2 would mean that it should be 2 times wider that the first or third one.

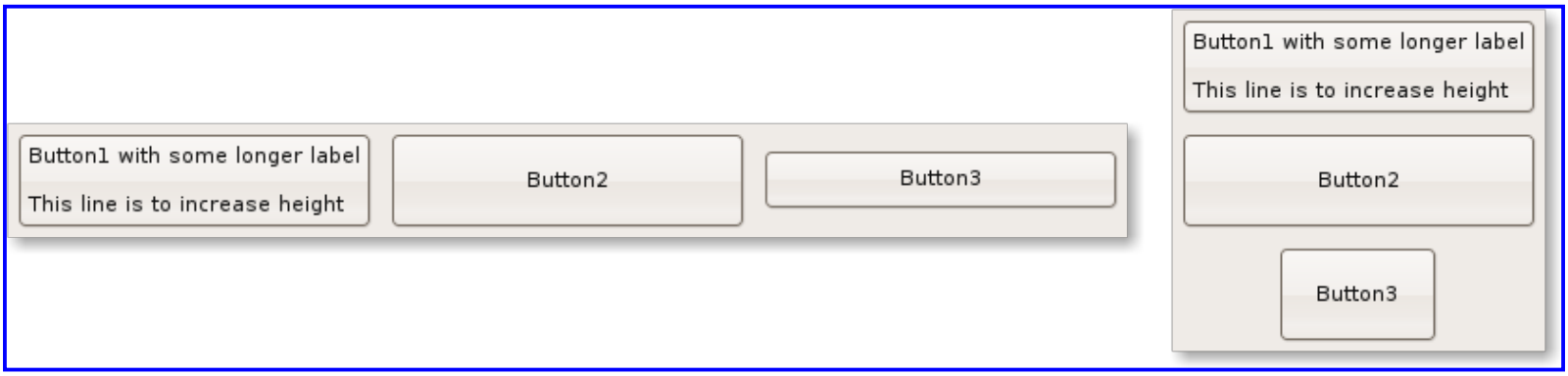
Genreally you can set any value you want here. One of common techniques is to set percentage value - for example setting proportions to values 20, 30 and 50 would mean that the first button should occupy 20% of the space, the second one 30% and the third one 50%. The special value 0 means that this item should be skipped in proportion calculations and it will won't automatically adjust it's size when main window is resized.

So if we want to get 1:2:3 proportions, we set 1 for Button1, 2 for Button2 and 3 for Button3. And here is the result:



Also note that if you create resizable window and resize it, those 3 buttons will automatically grow and keep proportions (when you set Proportion to 0, such item won't grow).

The other dimension which is not controlled by Proportion property may be controlled using "Expand" and "Shaped" properties. They are simple checkbockses. If you check the first one, the item will expand it's width/height and will get the size of the biggest item managed by the same sizer. Selecting "Shaped" will also make the item resize but such item will try to keep the initial proportion between width and height so it don't have to use all available space. The effect of "Expand" and "Shaped" properties can be seen when items differ in width/height, here's the example:

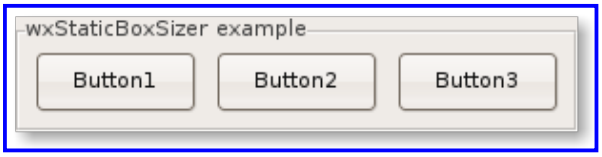


In this examples, the first button is the one that has the biggest size of all three buttons. It's size will be used to adjust sizes of other buttons. The second one has "Expand" flag enabled and the third one does not. What can be seen here is that the third button did expand only in one dimension (the one managed by "Proportion" property), the second dimension was left untouched.

You may also note that when item does not fit the entire area, it is centered. That's default behavior which can be changed. In properties of components managed by sizer you will find properties called **Horizontal align** and **Vertical align**. Using these properties you can select where the item should be located when it's smaller that it's available area.

[edit] wxStaticBoxSizer

This sizer is simillar to wxBoxSizer with one exception: it adds extra static box around managed items:



[edit] wxGridSizer

This sizer is little bit more advanced than wxBoxSizer because it does align items on grid, not on one line. You can specify number of columns or number of rows in properties "Cols" and "Rows". By default number of columns is set to 3 and number of rows is set to 0. When 0 is used, it does

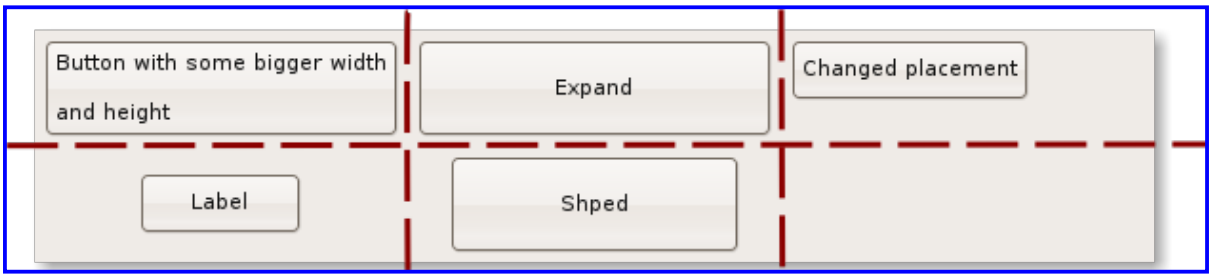
mean that sizer should automatically calculate this value to keep all managed components inside.

Also note that there's one assumption in wxWidgets - all cells must have same width and all must have same height. This mean that the tallest managed item will set the height of all rows and the widest one will set the widthts of all columns. If item won't use whole cell, it will be surrounded by empty space.

In propreties of wxGridsizer you can also find horizontal and vertical spacings. These values set the spacing added between items. The result of using them is simillar to using borders around managed components but with borders you would have to set borders for all items separately.

In case of wxGridSizer, the "Proportion" value is not used. But you may use "Expand", "Shaped" or placement properties to adjust the result.

Here's an example of grid sizer:



[edit] wxFlexGridSizer

This is one of the very usefull one. It works simillarily to wxGridSizer but it does not force all columns/rows to have same width/height. Also not all comuns do resize when there's more space available for the sizer. Comparing to wxGridSizer you will find two extra properties here: "Growable cols" and "Growable rows". In these properties you may provide list of columns / rows which should automatically resize (very usefull to handle resizing of the window) by providing list of numbers separated using coma (,). Numbers should start from 0, so if you would like second and third column to resize, put "1,2" into "Growable cols".

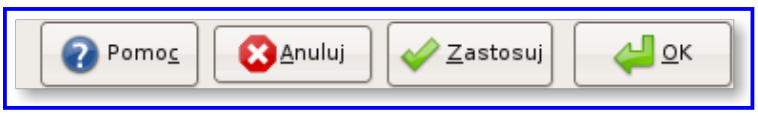
Unfortunately you won't be able to set proportions other that 1:1 ("Proportion" property is also not used in this sizer).

[edit] wxStdDialogButtonSizer

This is the specialized sizer with one purpose - provide set of standard buttons with respect to platform's look and feel. This sizer has predefined list of components which it can handle - all are buttons and you can manage them using sizer's properties. You can not add your custom items into this sizer.

Each button on this sizer has two properties in sizer - whether button should be enabled (for example wxID_OK for OK button) and it's label. Note that labels for most buttons will be generated automatically and wxWidgets will skip them when it would be able to read labels from system settings (for example on linux only the Context Help button does not have label). It's also very probable that labels will be provided with language-specific strings (depending on current system language). Not all configurations of buttons are valid - they won't break the application but buttons will overlap (for example Yes and Ok).

Here's and example of the sizer (note that labels are in Polish and do mean: Help, Cancel, Apply and Ok):



This screenshoot was taken on linux box. You may see that buttons have some extra image which is not available for standard buttons. With such layout, users of the application will feel more comfortable and such 'small' things make users say that the GUI was designed very well ;). On windows the sizer would have same buttons but without images and they would be in different order - the one that is used on windows by default.

All those sizers can be found on the "Layout" pallete page. You can find two other items there which are not sizers but can be used for layout

purposes.

The first one is Spacer. It can be added to other sizer when you simply need some empty space instead of any particular component. Spacers have only width and height.

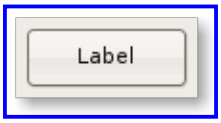
The second one is wxSplitterWindow. When you use this component, you can add one or two child components into it and they will be separated with dividing line (it's called sash in wxWidgets). On the application you can drag it to dynamically adjust the size of managed items.

Now since we know something about sizers let's talk about other components.

[edit] Usefull components

wxWidgets provides all standard graphical components like static texts, buttons, edit boxes - all those are supported in wxSmith so you can build a nice working environment. I'll shortly describe those that are really usefull in standard applications.

[edit] wxButton

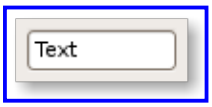


This is standard button, very common component. Its main purpose is to fire some event when you click on it and that's it.

[edit] wxStaticText

This item only shows user-defined text on the window. It don't generate any events.

[edit] wxTextCtrl



This item let's user enter some text. It's basic functinality allow to enter one-line text. After changing it's properties it may become more advanced multiline editor with support for different fonts, colours and text styles.

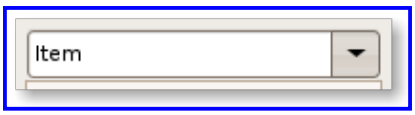
[edit] wxPanel

Although this item may be separate resource, it can also be added as other resource's component. This item may be used as background for other components. It has few functionalities - it can be used as background in notebooks and it may be a place where you put items without sizers. But it also has few drawbacks - wxWidgets tends to do some tricks with wxPanel's size so it's not always possible to set size we really want.

[edit] wxChoice

This item provides drop-down list with available options. It works simillar to wxComboBox but you can not enter your own value. It can be used in situations when user should use one of available options. Event is generated when user change selected item.

[edit] wxComboBox

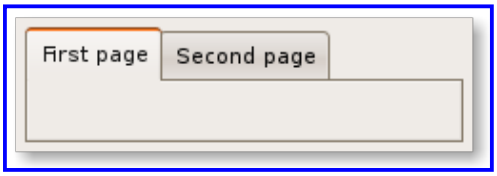


This item provides drop-down list simillarily to wxChoice. In case of this item, user can select one of available values or enter his own if the item is not in Read-Only state. Very good example of it's usage is to provide some text entry with stored list of previously entered values (like in search boxes). Simillarily to wxChoice, changing selection generates event. Additinoally changing the text and pressing enter also generates events.

[edit] wxListBox

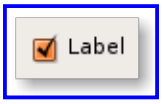
This item provides list where you can select one or more items. Changing item or double-clicking on it generates event.

[edit] wxNotebook



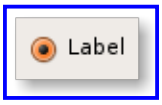
Notebook is very usefull for complex resources since it groups content in form of tabs. Addintionally wxChoicebook and wxCombobook are also available but are not described here. Changing selected notebook generates two events: one when the page starts to change (before change) and one when the page finished changing (after change)

[edit] wxCheckBox



Check box may be used to map boolean values, additionally 2-state box can also be generated (with values: yes/no/unspecified). Changing value emits event.

[edit] wxRadioBox



Radio box works simillarily to check boxes. The difference is that only one radiobox in group can be selected. Groups may be defined by using wxRB_GOUP style - radiobox which have this style set starts new group. Changing selected radiobox emits event.

[edit] wxGauge

WxSmith tutorial: Building more complex window

From CodeBlocks

Jump to: navigation, search

Contents

[edit] Building more complex window

Hi, in this tutorial I'll sow you some example of more complex window. Actually it would look complex when you start working with wxSmith. After writing few applications you will probably build few resources with this or even higher complexity so don't be scared. I'm working with wxSmith almost every day (it's really the best way to find bugs and discover nice improovements and that's what I should do as it's creator ;)) and although it may look comlex at the beginning, you can quickly get comfortable with it.

I also want you to know that as I'm writing the beginning of this tutorial, I'm also beginning of writing the application - I didn't prepared ready solution since I would like to show process of creating window in natural way (maybe even with some mistakes), so while writing the tutorial I'll also create the application.

We will start with empty frame - if you've read previous tutorials you shouold create one without

problems.

[edit] First task - some basic concept

My proposition for exmple application would be something to manage colleciton of CD-roms. So at the beginning we should realize what should be presented at the main window. Let's say that we would have most informations at the main window (although it may not be a very good assumption, it would help to show creation of more complex resources).

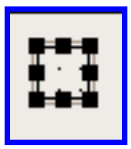
I would like to have some list of CD-roms on the left and some details of selected CD-rom on the right. So basically we have two regions in our window: one with the list and one with the details. I'd like also to use sizers because the application should be cross-platform and main window should be resizable.

[edit] Building skeleton

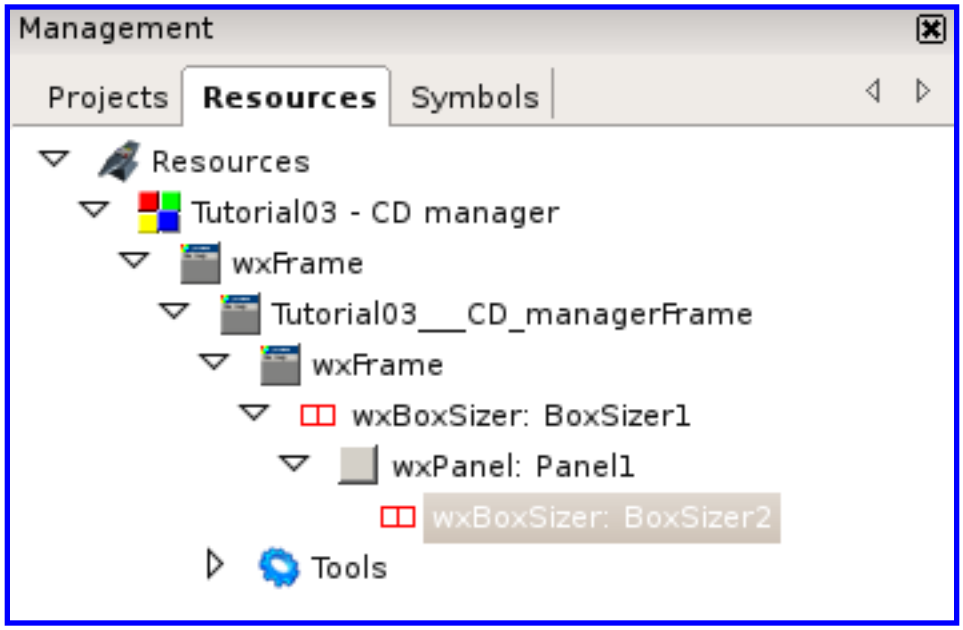
If want the application to look nice on both windows and linux, we should do some trick presented in the first tutorial - we had to add extra wxPanel into frame to make the background better. Let's to the same thing here:

- First add wxBoxSizer directly into resource
- Add wxPanel into that sizer
- Set size of border to 0 to make the panel cover entine window

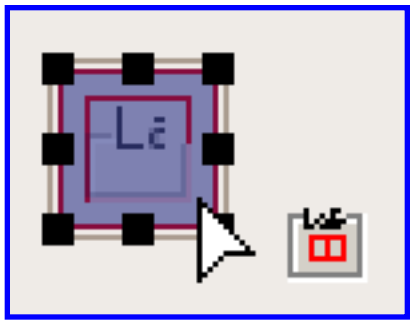
Now we should have some this result:



Now let's add wxBoxSizer into it - this sizer will manage out two main regions. After that the tree structure should look like this:



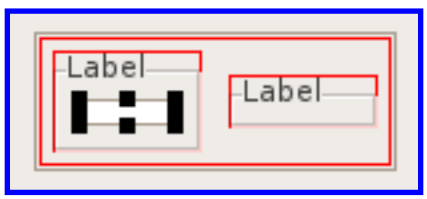
Now we can start filling those two regions - each one may have only one item (so the sizer we've just added will manage them). Usually in such sutiatuion I use wxStaticBoxSizer because we can mark regions and put some name for them. So let's add two wxStaticBoxSizer-s into the window. Be carefull while adding the second one because you may add it into the first region - always remember that parent for new items is always the item totally covered with blue:



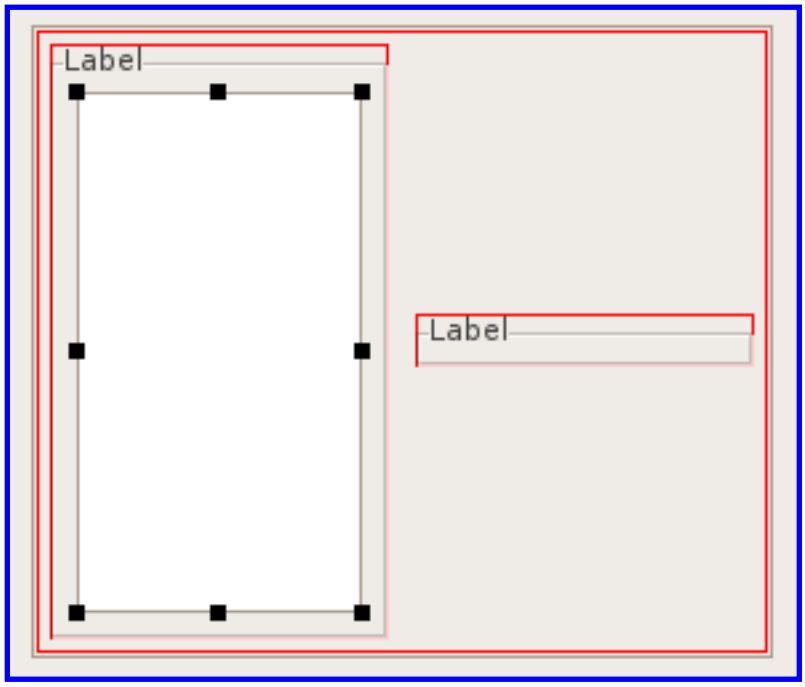
To add second sizer properly you must click on the border surrounding the first one just like on the picture. Note that now the resource is small but it will change when we will add some items into regions.

[edit] Building the List of CDs region

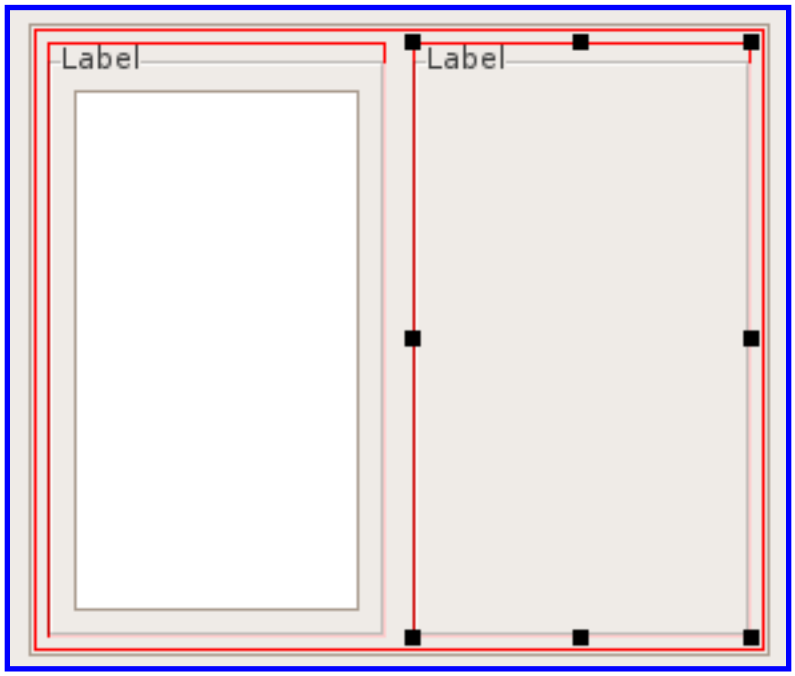
Ok, now let's add some item which will show list of CDs. We can use wxListBox here:



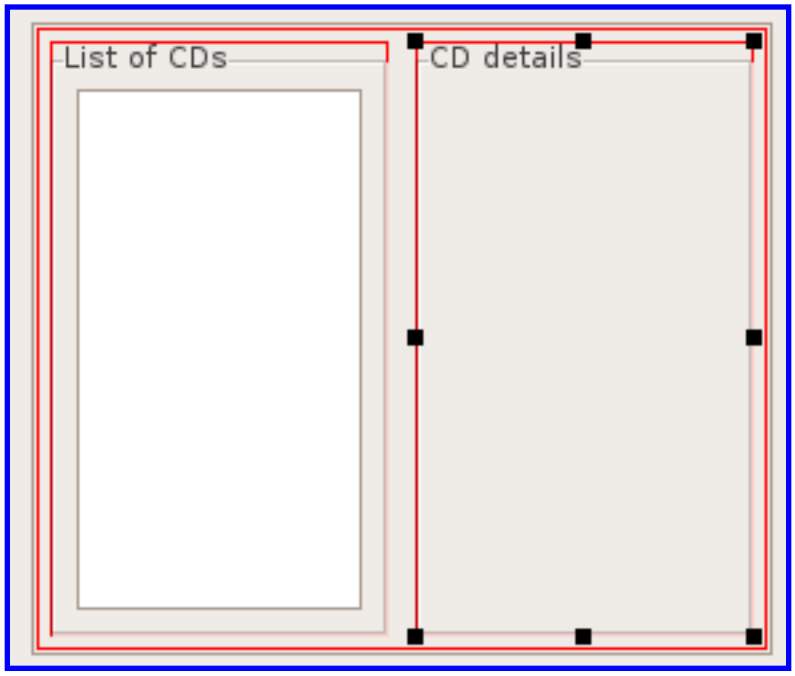
The list is rather small so let's resize it using drag-boxes to something bigger and higher:



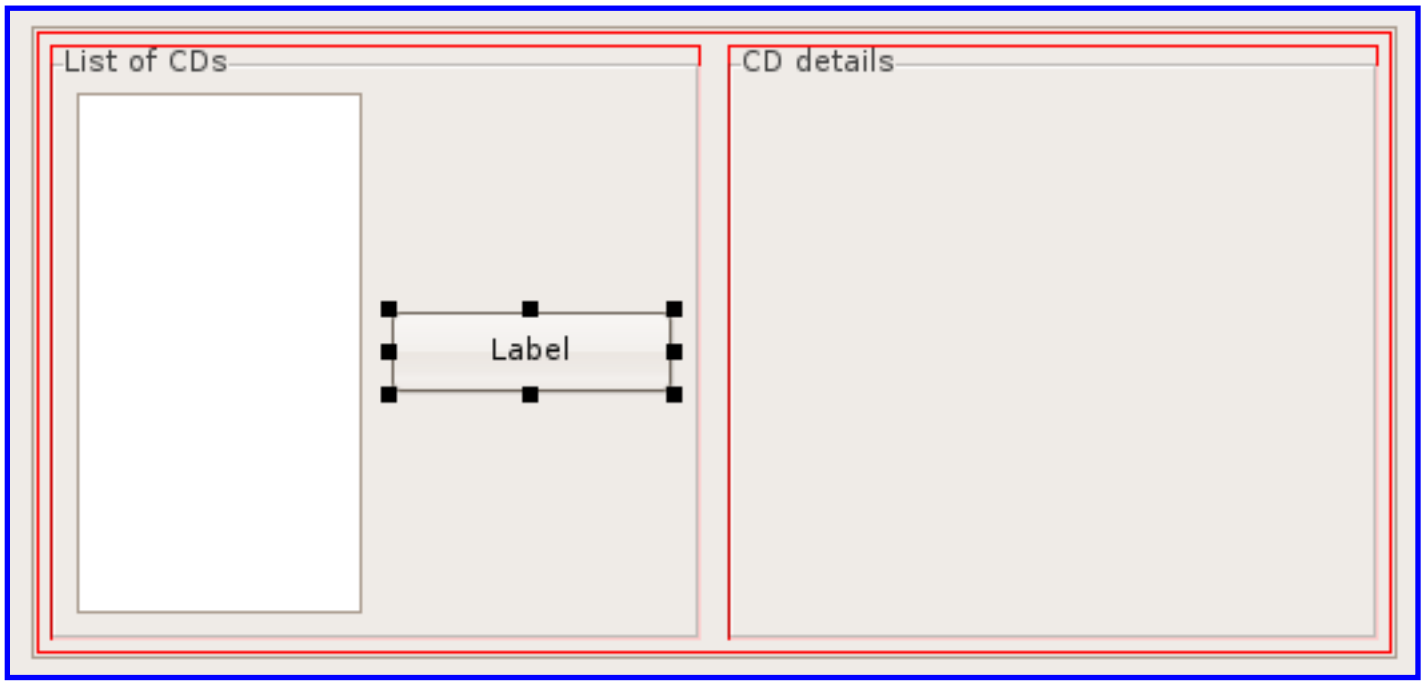
Ok, but the second sizer did not resize. That's right since we didn't turn on it's **Expand'** ***flag. To do this let's click on the second sizer and check it's*** *Expand* property:



Now we can change labels of regions since they are fully visible now. To do this click on each region's sizer and change the **Label** property:

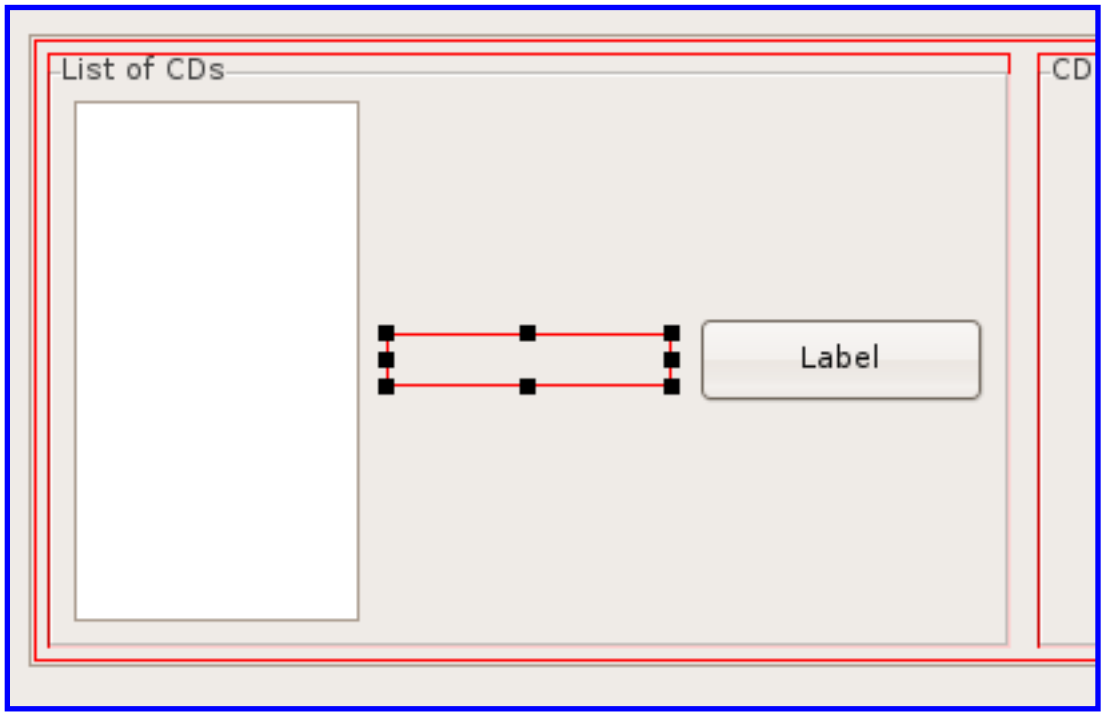


Ok but we would like to have ability to add and delete selected cd - we will insert two buttons for this purpose in the first region. To add the button choose it from palette and add into the sizer. Note that we can now point wxListBox item - it can not have children so wxSmith will try to add new item before or after it. After adding button we would have such result:

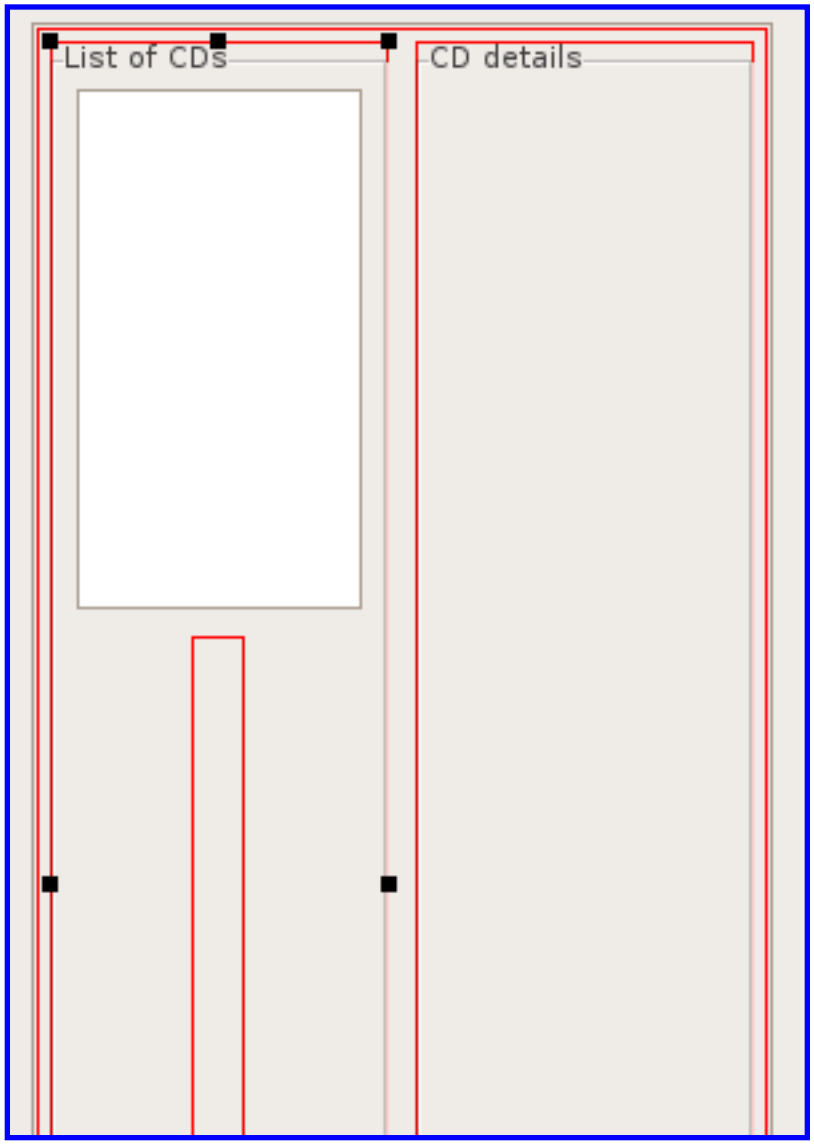


Hmm, the button should be under the list, not next to it. wxStaticBoxSizer did align items as it should - by default it aligns them in horizontal line. To change the direction to vertical, you can change the **Orientation** property of wxStaticBoxSizer. But before we change it let's predict one more thing. I would like to have two buttons instead of only one and I'd like them to be aligned horizontally. Since wxStaticBox will be changed to vertical mode, we will have to use another sizer used only for buttons.

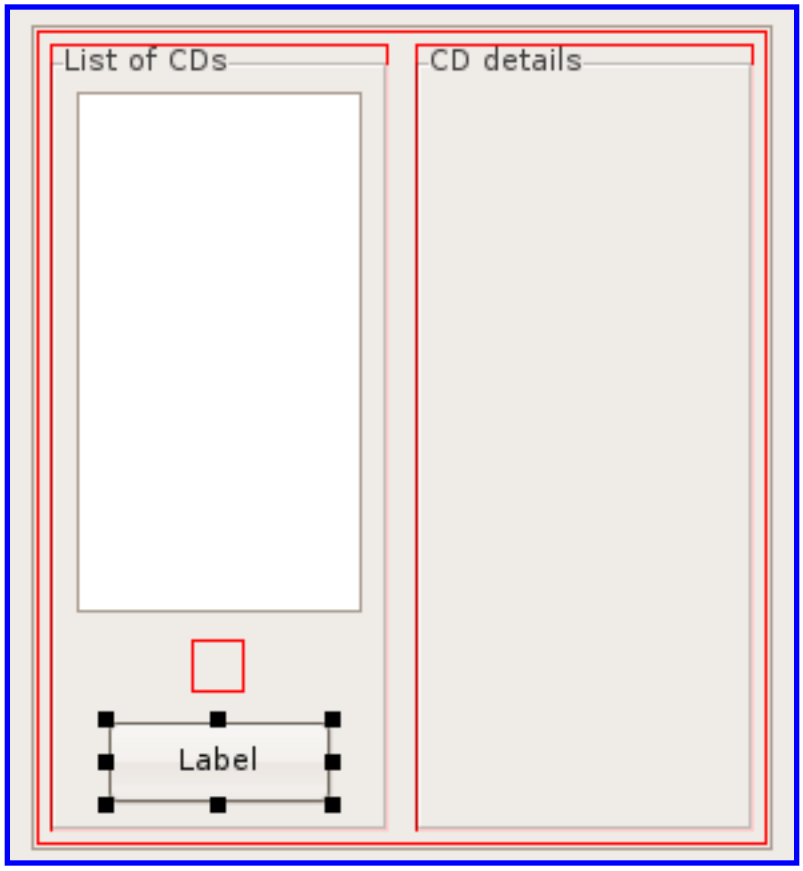
So let's add wxBoxSizer right after the list and before button:



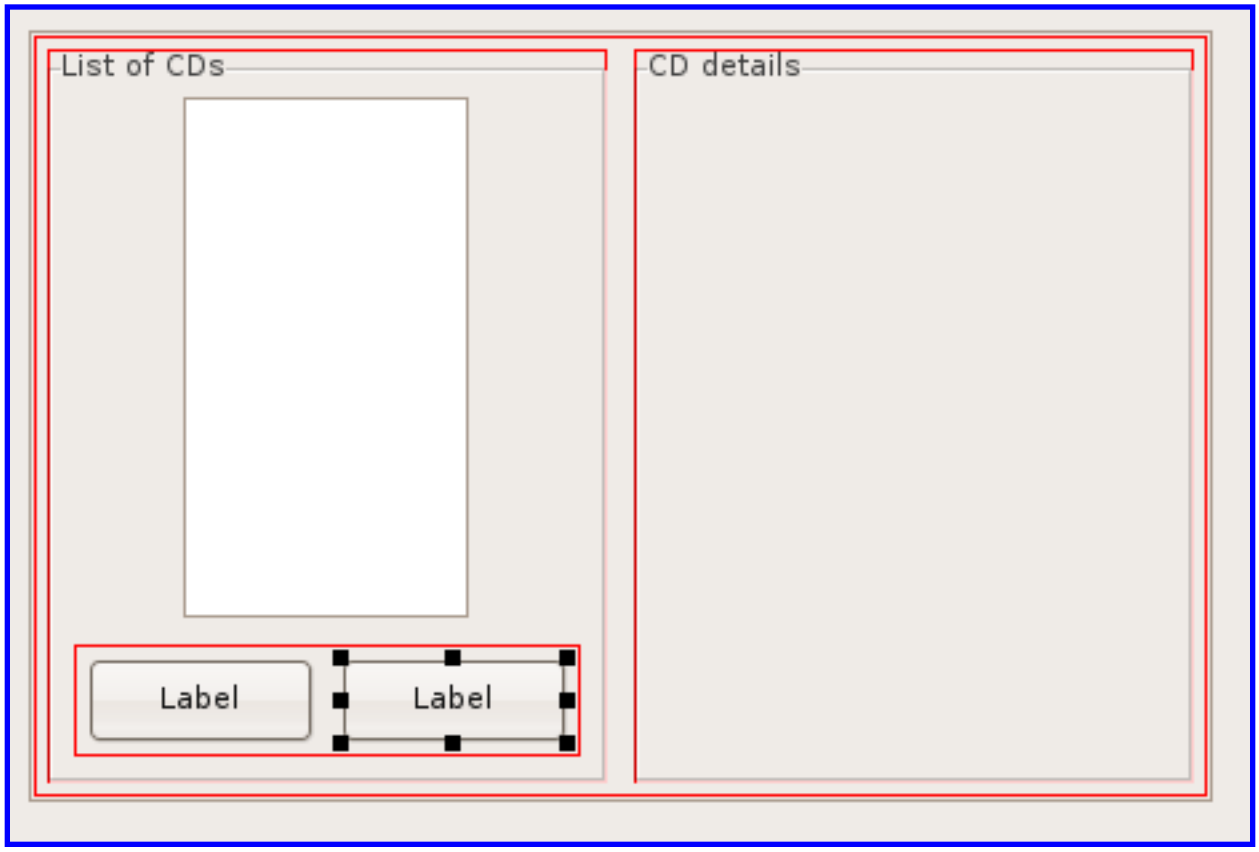
Now let's change **Orientation** of the wxStaticBoxSizer to vertical. After changing we have following result:



We can see that something's wrong here. The wxBoxSizer is abnormally stretched and if we scroll the editor we can see that the same goes to button. We said in previous tutorial that wxBoxSizer an wxStaticBoxSizer are trying to keep same size of items in one direction - that's what happened here. wxStaticBoxSizer used the highest item (list of CDs) and applied this height into all managed items. How to disable it? By setting **Proportion** property of both wxBoxSizer and wxButton to 0. This will notify that those items shouldn't be used in height calculations:



Now let's move the button into the wxBoxSizer - you can very easily do it by clickong on the button and dragging it into the sizer. After that we can add second button next to the first one:



Nowe we can see that the list don't have **Expand** property set (otherwise it would be almost as wide as the region.

Now I'd like to see how current work does look in the application. We could simply run it (F9) or by pressing the preview button on the right part of wxSmith editor:

(If you run application you will see that there's menu and statusbar, on the preview it won't show, wxSmith still misses few features ;))

Our window looks good, not perfect but let's leave some polishing for later and fill the CD details region.

[edit] Building CD details region

Inside CD details I'd like to have description of CD:

- Type (Music/Program/Backup/Other)
- Title

- Artist/Author
- Date of release
- Description

For such list we would need some grid sizer. Ok, but the region has wxStaticBoxSizer which align items only in one line. To overcome this limitation we can just add another sizer into wxStaticBoxSizer and use new one as the grid. From two available sizers I suggest wxFlexGridSizer (wxGridSizer would force all cells to have same size which wouldn't look good because row with description may need some more space than others). So let's add new sizer:

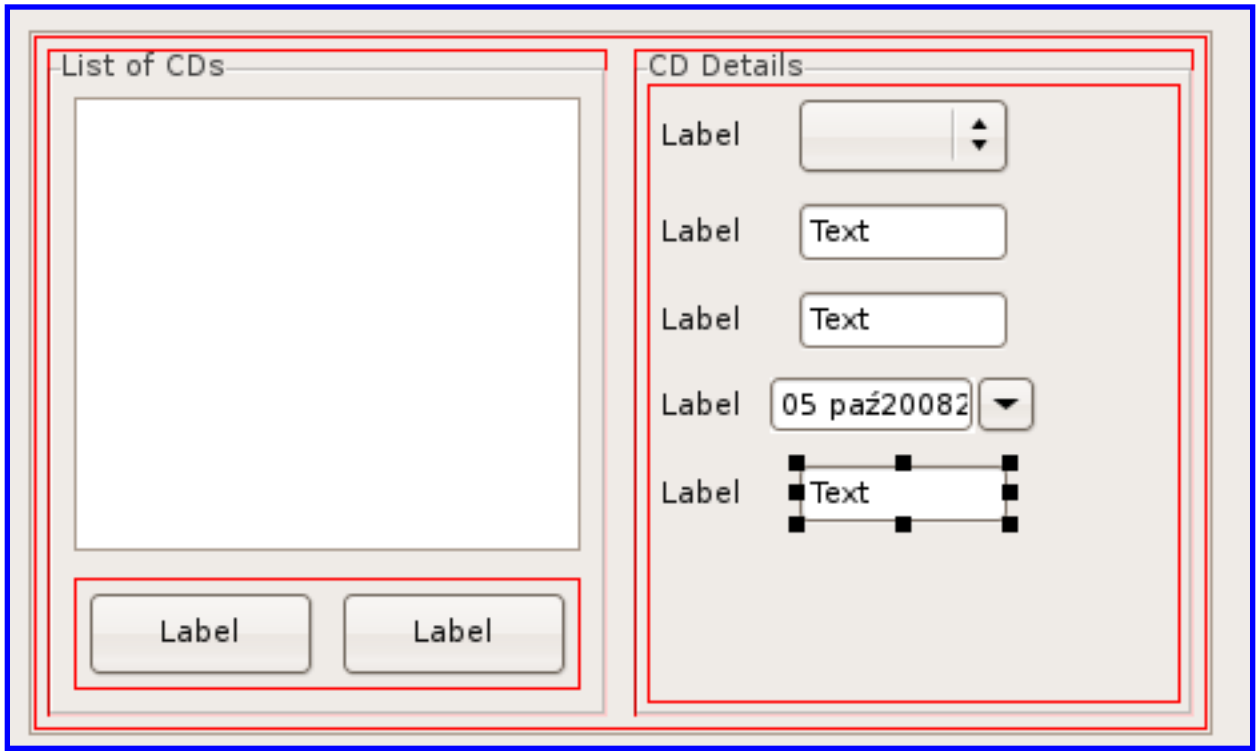
- Add wxFlexGridSizer into wxStaticBoxSizer
- Check **Expand** property of new item
- Set **Border Size** to 0 to avoid some unwanted border.

[edit] Adding items

Data presented here should be shown in two columns - first for the label and second for value so let's change the **Cols** property to 2. And now it's time to add the content. For labels we will use wxStaticText and for the value we will use different items:

- wxChoice for Type
- wxTextCtrl for Title, Artist/Author and Description
- wxDatePickerCtrl for Date of release

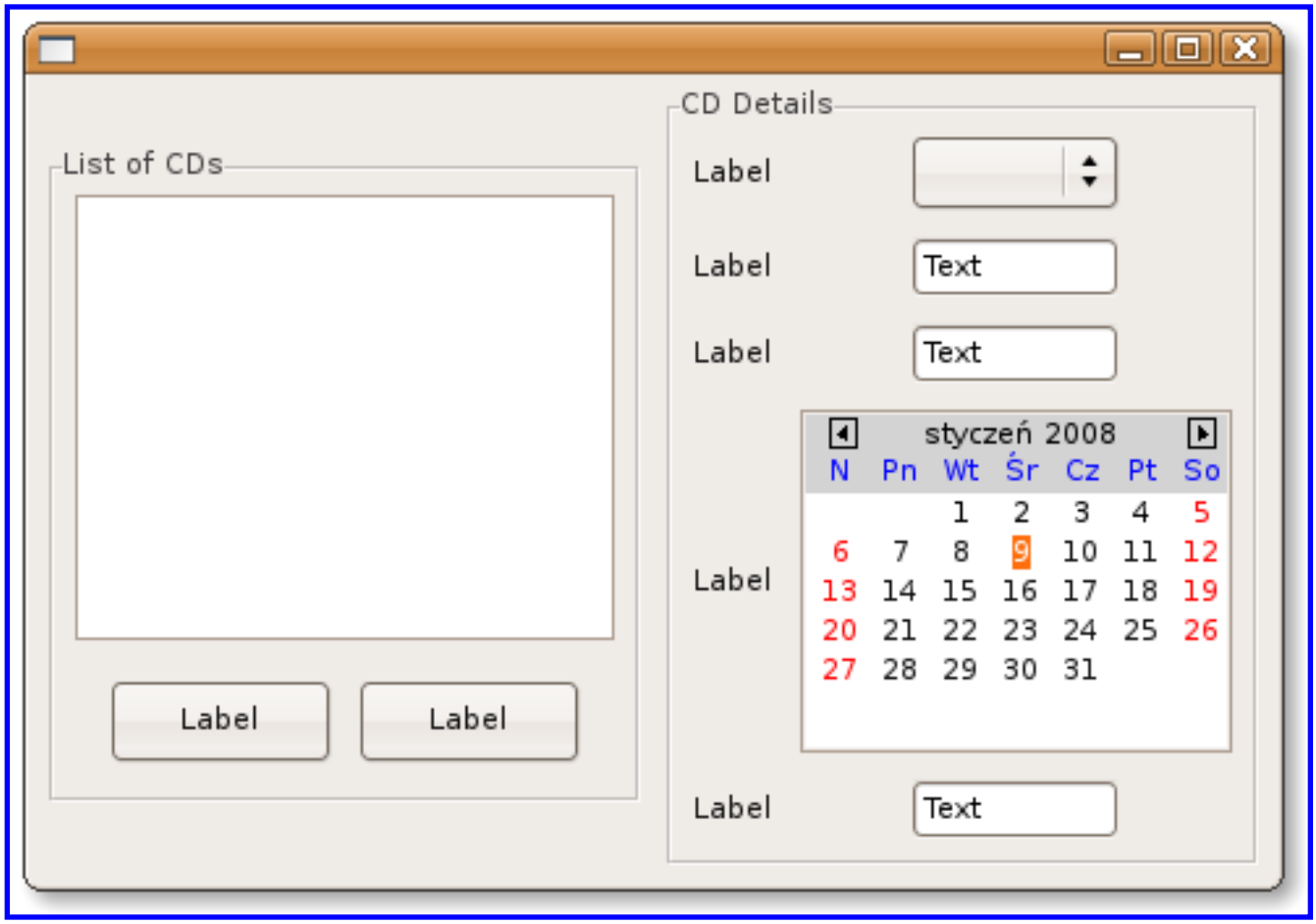
After adding items we should have following view:



Hmm, as I've said at the beginning - I didn't prepared anything before writing this tutorial. And guess what. It looks like wxDatePickerCtrl is broken on linux and sometimes crashes C::B (I didn't check on windows), don't know yet whether it's in wxWidgets or wxSmith - I'll have to investigate it. So we will have to change wxDatePickerCtrl to something else or remove it.

There's alternative for wxDatePickerCtrl - wxCallendarCtrl so first let's try to replace wxDatePickerCtrl. First we will have to remove wrong item (by using 'delete' key or by pressing 'X' button on the right side of the editor). After the change we can see that generated callendar is quite but according to other items in this region. We can shrink it a little bit by checking **wxCAL_SEQUENTIAL_MONTH_SELECTION** inside the **Style** property but it will make the item harder to navigate. We can additionally set **wxSUNKEN_BORDER** style to add some nice border around the callendar. Ok, not perfect but acceptable ;)

Now we have this screen:



[edit] Customizing and adjusting items

Ok, it looks like we have few things to do now:

- First is to check **Expand** property for the first region since (it didn't expand when the second region increased height)
- Second is to set proper labels for values in **CD details** region
- Third is to check **Expand** property for all items with values in same region
- Fourth thing is to adjust items used to edit values in **CD details** region

First three tasks are quite easy and I'll leave them to you :)

Fourth task will require few new things so I'll describe it better:

First let's remove text from text boxes - it's set by default but it would be better to have empty text by default. Text can be changed by modifying **Text** property so you should do this very quickly.

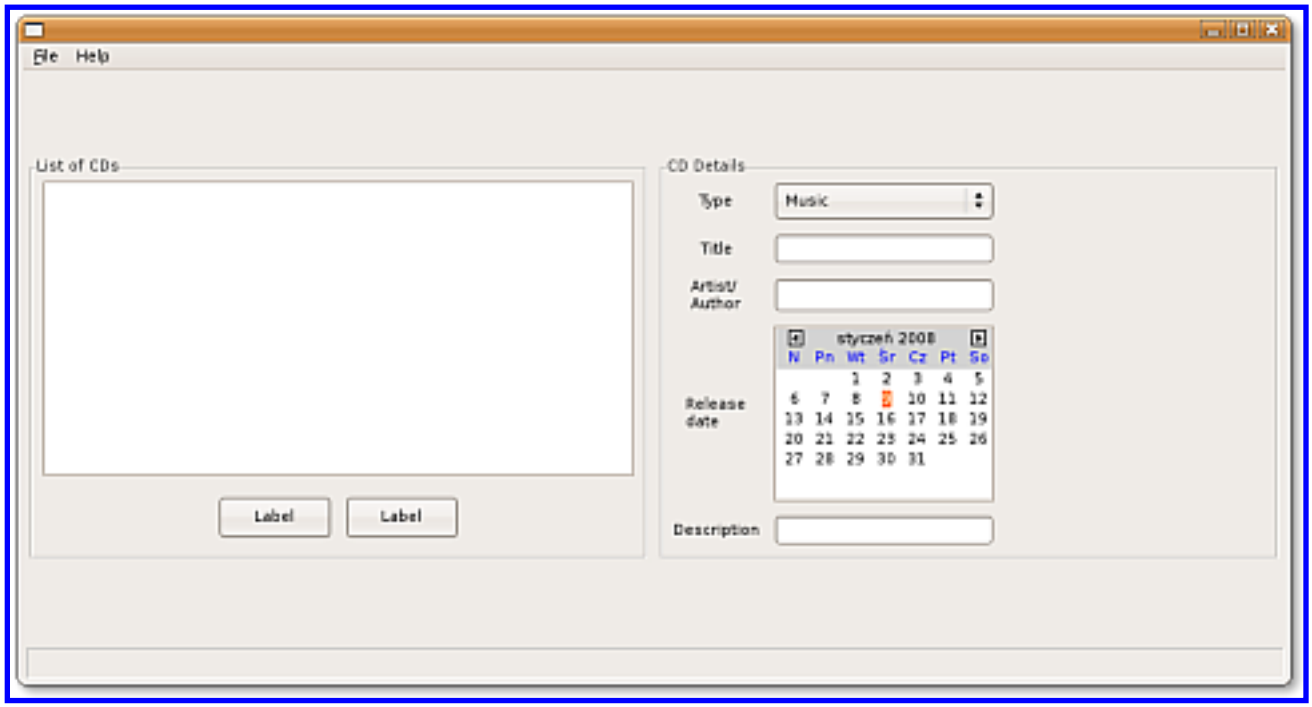
Now we have to set some values which could be choosen from **Type** value. It's done by editing **Choices** property of wxChoice. This is more advanced value consisting of few strings to entering it inside one line of text could be hard. Because of that, instead of text, there's a button on the right side. When you

click it, new window will pop-up where you can enter choices:

```
Music
Program
Backup
Other
```

[edit] Let's check how does the window resize

We have the first version of our window. But it's not the end of work. If we want it to be user-friendly, it should give ability to resize. We can check it by either showing preview or building and running application. The latter is preffered since it may produce better results. If the window will show, we should experiment by resizing it:



We can see two wrong things in this window:

- Regions did not expand vertically and stay centered
- Content of second region did not react for the resize

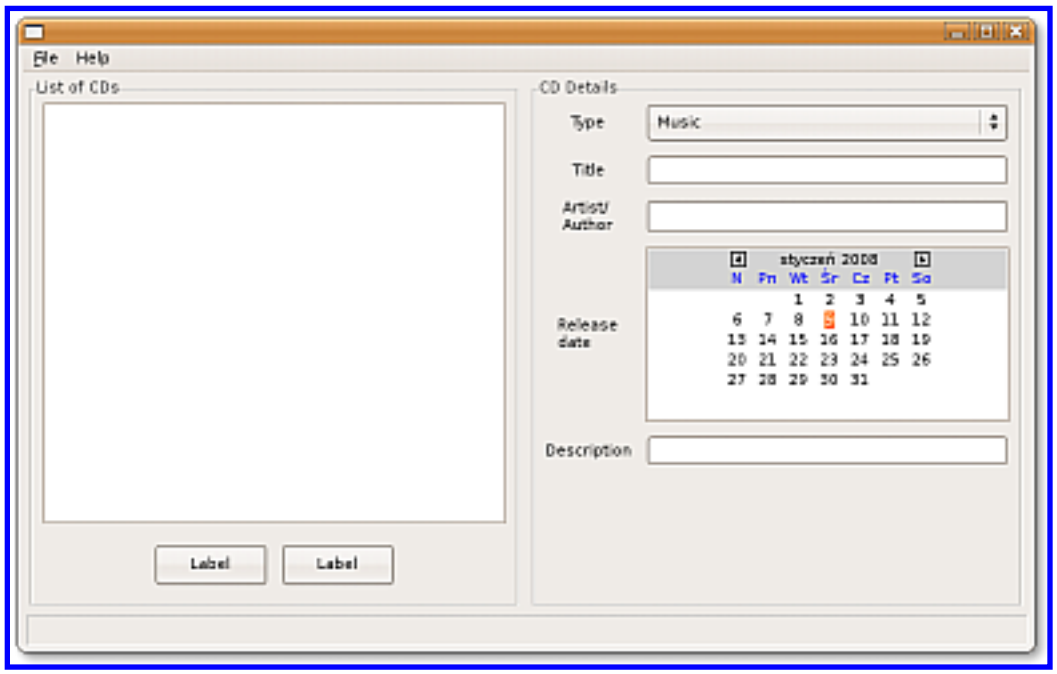
The first problem should be cuased by some missing **Expand** property so let's find it.

First we should check if regions have **Expand** property set: yes they have so this is not the problem. Generally in such situations we should continue checking in parent item - this is wxBoxSizer which always expand. Next there's wxPanel we used to make nice background and voila: this one does miss the

Expand property. So let's check it and test the result again and it works fine.

Now the second issue. Content inside **CD details** pane was based on wxFlexGridSizer. If you look into previous tutorial you may find out that this sizer has special property called **Growable cols**. Using this property we can set columns which should resize. I'd like the column with values to grow (we don't need more space for labels, dont we? ;)) so let's set the **Growable cols** to **"1"** (idexes are 0-based). Note that you may quickly find the wxFlexGridSizer by ooking into resource tree - we have only one such sizer in our project.

Now we have some more correct results:



[edit] Continue polishing the window

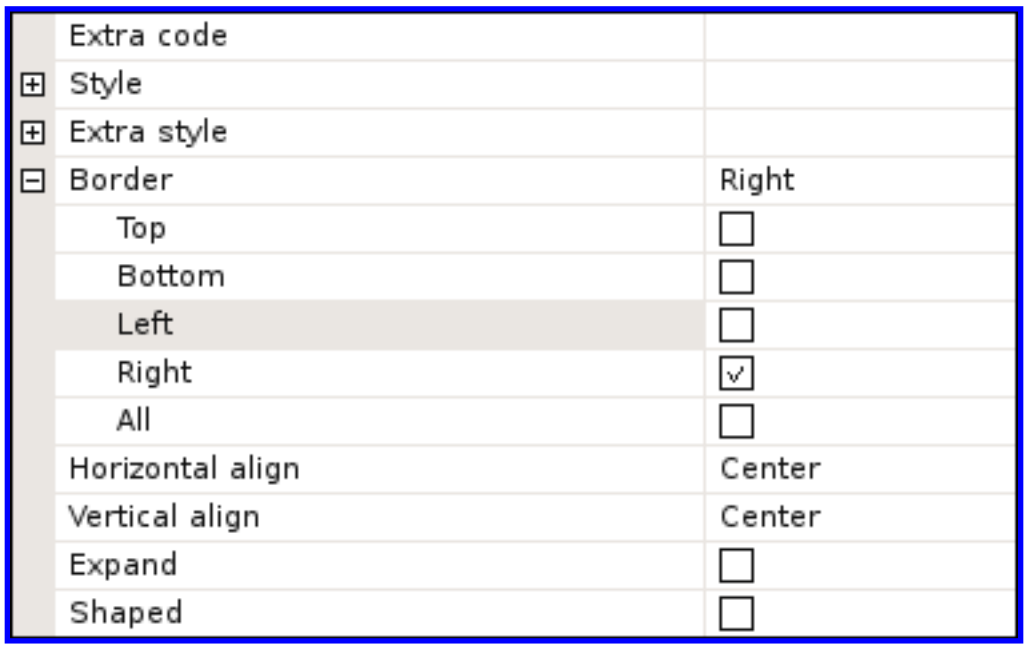
As I look into our window I see that I left few things on the first region. Buttons don't have new labels and there's to much space between them and the list so we can update those things right now.

As usual I'll leave changing the Labels to you - those buttons should be called **Add** and **Delete**. And now let's remove the wasted space.

The space is added because each item inside sizers may have extra border. By default it's set to 5 pixels so if we remove the border around buttons we would have more free space. But than the buttons won't have any space between them.

To overcome this problem we could use another property related to border also called **Border** (jay, I have to fix the naming before someone notice it) which expands to few check-boxes. By checking or

unchecking them we can enable or disable border at some item's edge. So if we remove top/bottom/left edge borders of the first button and top/bottom/right edge borders of the second button we would remove extra space but buttons will still have some gap between them. Here's how the property should look like for the first button:



[edit] Final words

Here we reach the end of this tutorial. Of course the window we've built is not in it's final shape and it could be adjusted - for example we could add some list remembering when and who borrowed the CD from us and who has it now, we could add some searching and so on. But to keep this tutorial quite short, I've decided to leave all those upgrades to you. I hope that you've learned something new and interesting here. Good bye and see you in next tutorial :)

WxSmith tutorial: Working with multiple resources

From CodeBlocks

Jump to: navigation, search

Contents

[edit] Working with multiple resources

Hello. In this tutorial I'll show you how to create application with more than one window. I'll also show how to invoke one window from another. At the end we will learn how to change man window in project settings.

So let's start.

[edit] Creating empty project

As usual we will start with empty project. You can find informations on how to do this in the first tutorial. Remember to crate frame-based application and select wxSmith as main RAD designer. Also let's name the project "Tutorial 4".

If empty application compiles and runs fine we can advance to next step.

[edit] Adding few dialogs

New resources can be added from **wxSmith** menu. You can find following options there:

- **Add wxPanel** - this will add new panel into resource
- **Add wxDialog** - this adds new dialog window
- **Add wxFrame** - this adds new frame window.

Ok, let's add new wxDialog. If you choose this option it will show following window:

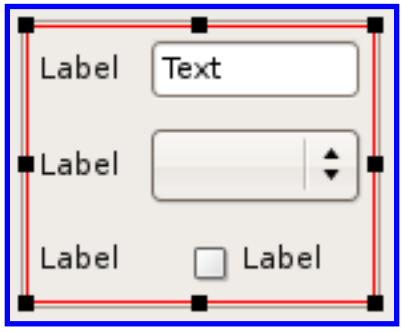
Here we can configure following parameters:

- Class name - it's name of class which will be generated for the resource. It will also be a name of resource
- Header file - name of header file with class declaration
- Source file - name of source file with class definition
- Xrc file - if you check this option you can enter name of XRC file which will contain XRC's structure (XRC files will be covered in other tutorial)
- Add XRC file to autoload list - this option is aailable only with XRC file and notifies wxSmith that it should automatically load XRC file when the application starts.

Now let's change name of resource to something better like "FirstDialog". Note that if you change class name, names of files are also updated (they are updated as long as you don't change them manually). After clicking OK you will be prompted for list of build target into which new resource will be added. Select both debug and release and we can nontinue.

New dialog is automatically opened in editor. Let's add some simple content:

Now let's add some wxFrame resource (FirstFrame) and add some content into it:



[edit] Using dialog window in modal mode

Dialogs can be shown in so-called modal mode. This means that when such dialog is shown, all other windows in the application are frozen as long as the dialog finishes it's job. Such dialogs are usefull when user should quickly provide some informations - for example the window used to configure new resource was modal.

Now let's try to invoke our dialog. Generally such dialog will pop-up as reaction for user action like choosing menu option or clicking on button. We will use he button since this approach is really easy.

Ok, let's switch into main frame - the one that was opened right after creating new project, and add sizer and button:

Now we have to create action for button-click event. The easiest way is to double-click the button in editor. wxSmith will automatically add new event handler function:

```
void Tutorial_4Frame::OnButton1Click(wxCommandEvent& event)
```

```
{
}
```

If you can't find this code, scroll down to the end of cpp file.

Now let's Invoke our dialog:

```
void Tutorial_4Frame::OnButton1Click(wxCommandEvent& event)
{
    FirstDialog dialog(this);
    dialog.ShowModal();
}
```

In the first added line we created dialog's object notifying that *this* window (main frame) is parent of dialog. In the second line we show the dialog. Note that call to ShowModal() blocks untill dialog is closed.

There's one more thing we need to add before our application compiles. Jump to the beginning of the file and add following code next to other includes:

```
#include "FirstDialog.h"
```

Note that you shouldn't add it inside code which looks like this:

```
//(*InternalHeaders(Tutorial_4Frame)
#include <wx/intl.h>
#include <wx/string.h>
//*)
```

This is block of code that is automatically generated by wxSmith. Every block starts with *//(*BlockName* comment and ends with *//*)*. You may find other blocks in both header and source files. If you change their content, all changes will be lost net time you change something in editor.

To sum things up, headers section should look like this:

```
#include "wx_pch.h"
#include "Tutorial_4Main.h"
#include <wx/msgdlg.h>
#include "FirstDialog.h"

//(*InternalHeaders(Tutorial_4Frame)
#include <wx/intl.h>
```

```
#include <wx/string.h>
//*)
```

Now we can compile and run our application- if you click button on main window, dialog will pop-up:

[edit] Using dialog and frame window in non-modal mode

Another mode used to show windows is modal-less mode. In such case new window does not block other windows in application and two (or more) windows can cooperate in one application.

Before we add new code into our application there's one more thing you should know. Each window may exist only as long as it's object (instance of resource's c++ class). So we can not use same approach as in case of modal dialog. In modal mode object was created as local variable of event handler. We could do this only because ShowModal method was blocked as long as diaog was shown. Now we will have to create objets using new opertor because objects must exist after leaving handler functin and also because windows not using modal mode will delete such objects automatically when window is closed.

To allow creating FirstFrame class we should add **#include "FirstFrame.h"** into list of includes just like in case of FirstDialog:

```
#include "wx_pch.h"
#include "Tutorial_4Main.h"
#include <wx/msgdlg.h>
#include "FirstDialog.h"
```

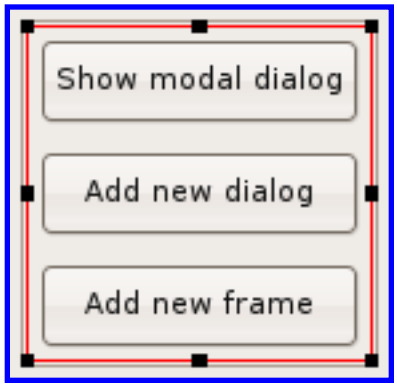
```
#include "FirstFrame.h"

//(*InternalHeaders(Tutorial_4Frame)
#include <wx/intl.h>
#include <wx/string.h>
//*)
```

Now let's add two more buttons to main frame:

- Add new dialog
- Add new frame

We can also name the first button to show what it does:



Now let's add handler to *Add new dialog* button:

```
void Tutorial_4Frame::OnButton2Click(wxCommandEvent& event)
{
    FirstDialog* dlg = new FirstDialog(this);
    dlg->Show();
}
```

Analogically we can write the code for firstFrame:

```
void Tutorial_4Frame::OnButton3Click(wxCommandEvent& event)
{
    FirstFrame* frm = new FirstFrame(this);
    frm->Show();
}
```

Now each time we click *Add new dialog* or *Add new frame* buton, new window shows up.

This is the end of this tutorial. I hope that you learned some new useful things. In the next tutorial I'll show how to use the last type of resources, wxPanel, inside other resources.

See you.

WxSmith tutorial: Using wxPanel resources

From CodeBlocks

Jump to: navigation, search

Contents

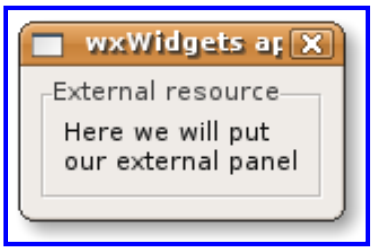
[edit] Using wxPanel resources

In some cases, one resource must be splitted into few smaller parts. Such split gives few advantages: it helps working more people on same project, it may give you cleaner view over really complex resources and it may help to divide source code for logical parts in case of few functionalities in one window. In this tutorial I'll show you how to create such resources.

[edit] Creating our playground

wxPanel resources can not live as independent items - they must be placed inside frame or dialog. We can use main resource created inside wizard. Assuming that you've read previous tutorials, creating simple window shouldn't be a big problem for you :).

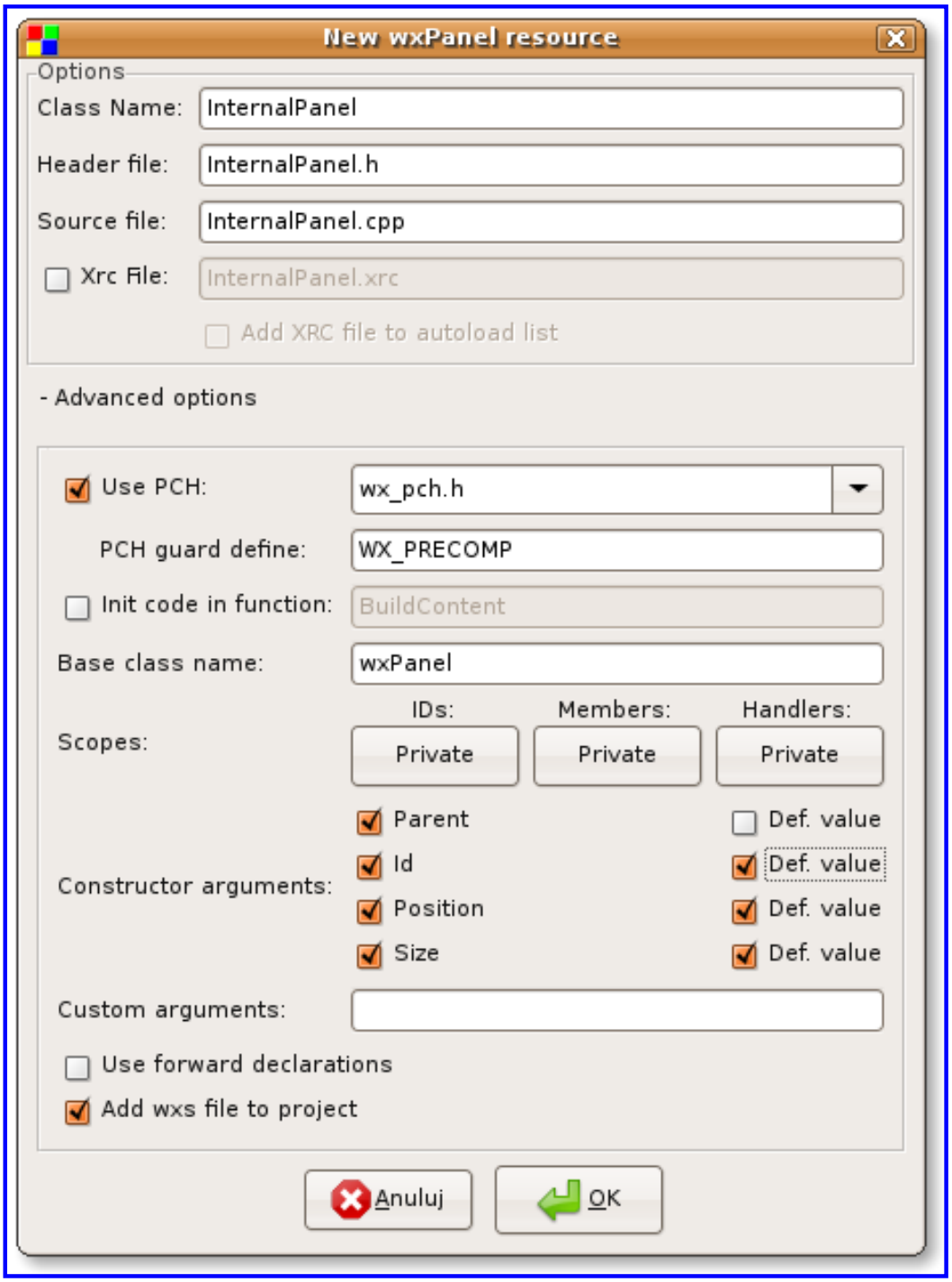
Let's start with something like this:



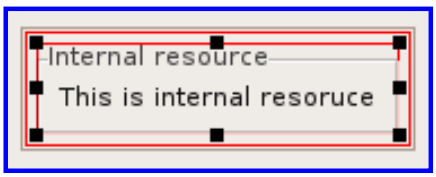
[edit] Adding wxPanel using "Custom" item

In this tutorial I'll show you two methds of embedding external wxPanel inside another resource. First one use "Custom" item which can be used to any kind of resource not know to wxSmith.

So let's create wxPanel resource. Note that used embedding method will affect initial configuration of resource. We want id, size and position of our panel to be controlled by parent resource so make sure that we add them into panel's constructor:



Now let's add some content into panel:



Final step is to add "Custom" item into main window and configure it properly. Custom icon is the last one in the "Standard palete" identified by this icon:

New item will be shown as black square with three question marks on the top. Let's resize it a little bit:

Now we need to adjust custom item's properties.

First thing we will adjust is "Creating code" property. As the name says, here we will be able to adjust the way wxSmith adds code responsible for creating this item. The default value is:

```
$(THIS) = new $(CLASS)($(PARENT),$(ID),$(POS),$(SIZE),$(STYLE),wxDefaultValidator,
$(NAME));
```

You may find that there are some macros used. They are here to help you map other properties of this item into ceating code:

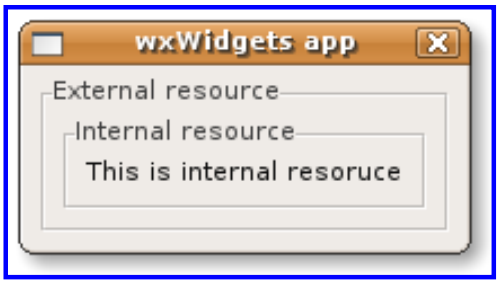
- $(THIS) is replaced by name of variable for this item
- $(CLASS) is replaced by name of item's class
- $(PARENT) is replaced by name of parent item (it's granted that it will be a pointer to class derived from wxWindow)
- $(ID) is replaced by value of Id property
- $(POS) is replaced by value of position property (it may be adjusted by using drag boxes)
- $(SIZE) is replaced by value of size property (it may be adjusted by using drag boxes)
- $(STYLE) is replaced by style property - since custom item doesn't have predefined set of styles, it's threated as normal string
- $(NAME) is replaced by generated name (which is equivalent to string representation of item's identifier)

The default code template creates standard item which uses default set of properties. Let's replace it by value corresponding to our panel's constructor:

```
$(THIS) = new $(CLASS)($(PARENT),$(ID),$(POS),$(SIZE));
```

Now we have to give name of header file with our external resource in the "Include file" property. I've created wxPanel with name "InterlanPanel" so header will be "InternalPanel.h". Also let's check the "Use "" for include..." since we're including local file.

The last property to adjust is the "Class name" - we need to put name of our panel here, in my case it's "InternalPanel". When we do this, our embedded resource is ready:



[edit] Adding custom panel through standard wxPanel

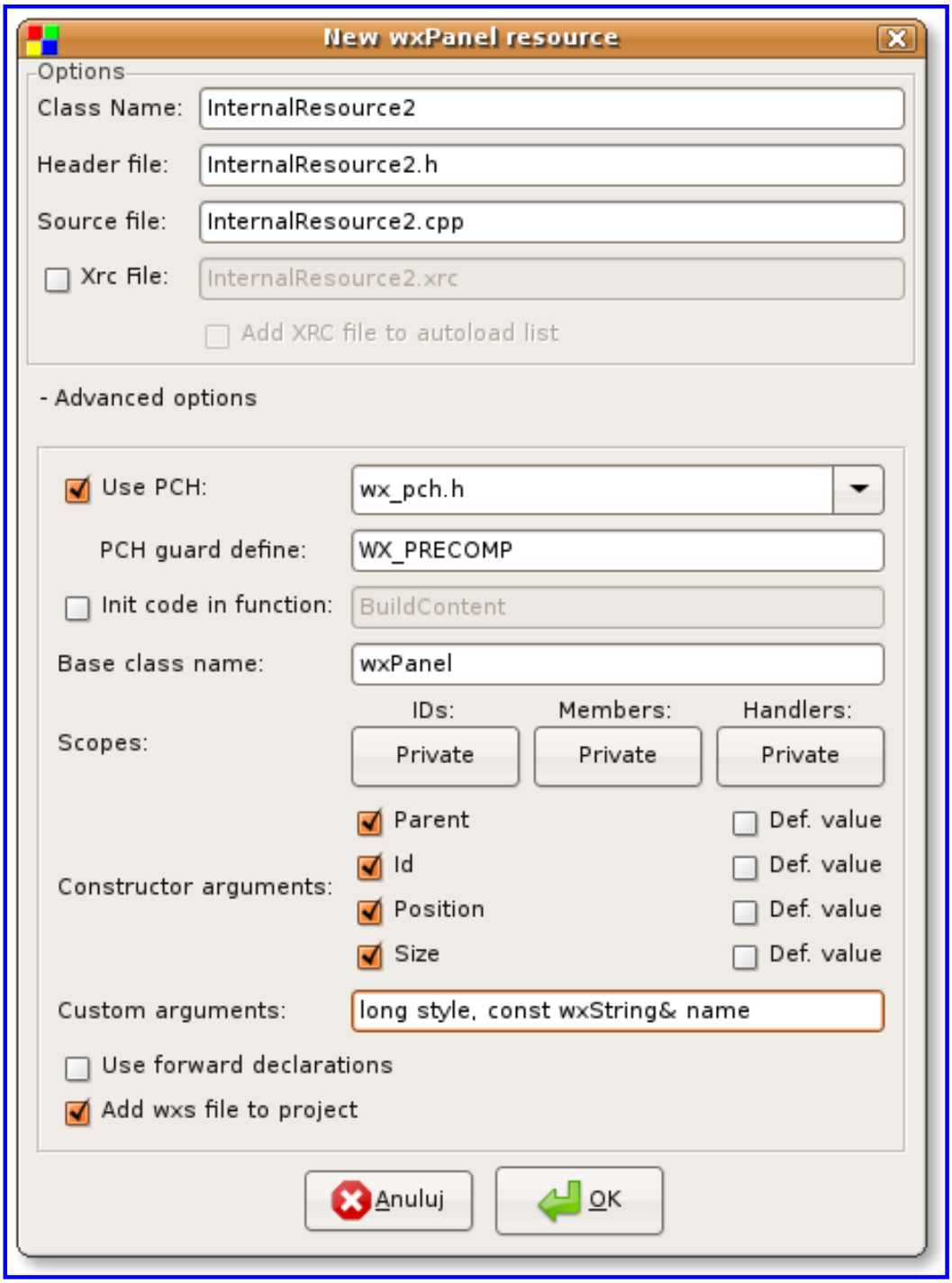
Secend way in which external resource may be used is through normal wxPanel. If you look closer, you'll find that most of items have property called "Class name" - this is the name of class which will be used as item's type. Bu default it contains name of original class in wxWidgets. Changing it will notify that different class will be used instead.

So to put our own panel here we can add "normal" panel into main resource and change class name to name of our internal resource. So far it's easy but there's one requirement to this technique - our internal resource must have exactly the same constructor arguments as in case of "original" class.

So let's take a look at the wxWidgets documentation and here's declaration of wxPanel's constructor:

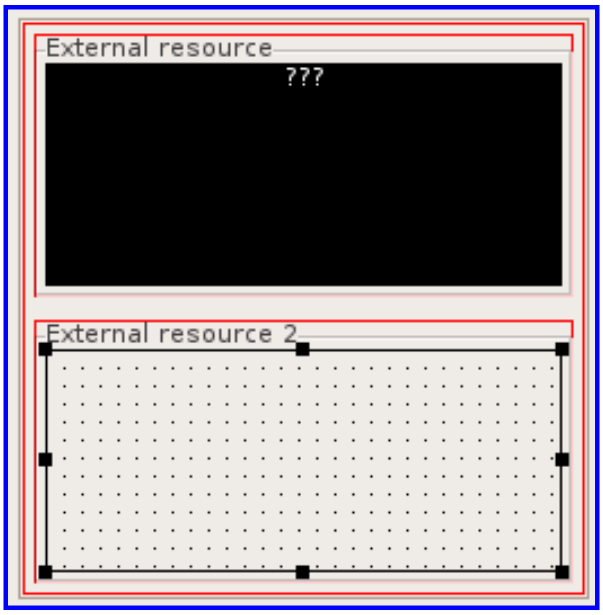
```
wxPanel( wxWindow* parent, wxWindowID id = wxID_ANY,
         const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize,
         long style = wxTAB_TRAVERSAL, const wxString& name = "panel")
```

Now let's create our own internal resource. We will have to adjust constructor arguments as in case of previous method:



Note that I've added custom arguments and turned off all default values (it's required since we can not easily add default values of our custom args).

Now that our panel is ready we can add it to main resource - let's add "normal" panel first:

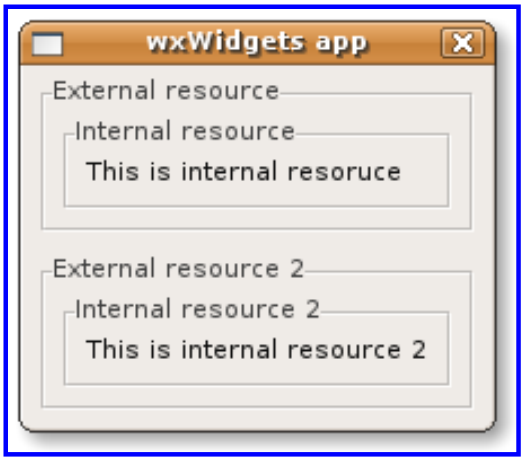


and change "Class name" property to name of our resource (in my case it's InternalResource2).

We still need to add #include "InternalResource2.h" into sources to make our project compile. In previous solution it was done automaticaly through properties. Now we don't have such system so let's add it manually (remember to put our include outside wxSmith's code section, otherwise any change in resource will remove our change). Note that we need to add it to header file of main resource:

```
//(*Headers(Tutorial5Dialog)
#include <wx/sizer.h>
#include <wx/panel.h>
#include "InternalPanel.h"
#include <wx/dialog.h>
//*)
#include "InternalResource2.h"
```

Now we can run our application:



I've presented two easiest methods of creating complex window from smaller resources. Perhaps you could find other ways to do it (if you do so, you can extend this tutorial :)). On wxSmith's wishlist there's also nice feature request to allow adding

external panels int more natural way (just by few clicks) which will probably make other methods obsolete. But I hope that I gave you enough informations to build nice compound resources so far :)

WxSmith tutorial: Accessing items in resource

From CodeBlocks

Jump to: navigation, search

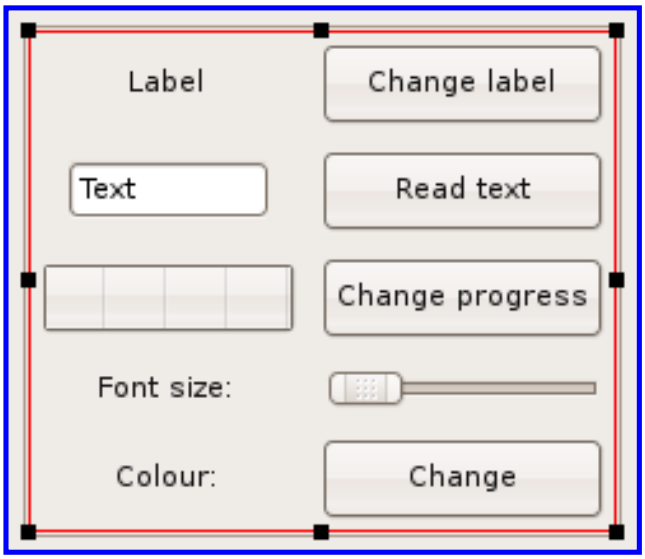
Contents

[edit] Accessing items in resource

Wecome in next tutorial. This time I'll show you how to access items in resource. I'll show you some basics - for example how to read data from text boxes, change labels, and some more advanced things like changing colours and fonts while application is running. As usual we will start with empty application. You shouldn't have any problems with it - all instructions are in first tutorial.

[edit] Adding few items we will work on

In this tutorial we will work on items so let's add some:



In this resource wxFlexGridSizer with 2 columns is used as background. In first three rows we have wxStaticText, wxTextCtrl and wxGauge - we will operate on them using buttons on the right column. Below there is wxSlider which will be used to change size of font - I've changed it's **Max** property to 30 to limit size range. Below there's button which will be used to change colours of some items.

[edit] Accessing items through members of resource class

If you add item in editor, in most cases wxSmith will add new member variable into resource's c++ class. All those members are listed inside `//(*Declarations ... //*)` code blocks. In case of window we crated it should look like this:

```
...
class Tutorial_6Frame: public wxFrame
{
    ...

        //(*Declarations(Tutorial_6Frame)
        wxSlider* Slider1;
        wxButton* Button4;
        wxStaticText* StaticText2;
        wxButton* Button1;
        wxGauge* Gauge1;
        wxStaticText* StaticText1;
        wxStaticText* StaticText3;
        wxButton* Button2;
        wxButton* Button3;
        wxStatusBar* StatusBar1;
        wxTextCtrl* TextCtrl1;
```

```
        //*)
    ...
};
...
```

Each item which has such member variable will also have following properties:

- **Var name** - name of used variable
- **Is member** - switch whether this item should be accessible through class member variable

So if you want to change name used to access item, **Var name** is right property to change.

wxSmith forces few restrictions on variable names. Most obvious is that each variable name must be valid c++ identifier so you can not provide special characters or even spaces. Another limitations is that variable names must be unique. If name is invalid, wxSmith will automatically replace it with name that matches criteria.

[edit] Changing label in wxStaticText

Ok, let's do some basic excercise by changing label of first wxStaticText. To do this double click on button next to it to generate event handler and change the code to following thing:

```
void Tutorial_6Frame::OnButton1Click(wxCommandEvent& event)
{
    StaticText1->SetLabel(_("Label changed"));
    Layout();
}
```

In first line of function body we used **SetLabel()** function to change text of label. Because length of the text changes, we must recalculate positions which is done by calling **Layout()** function.

In case of static items (those which can't be changed by user) usually have two functions: GetLabel and SetLabel which read and write content presented by this item.

You may wonder why we used some weird notation for string:

```
_("Label changed")
```

instead of simple

```
"Label changed"
```

There are two advantages of such approach:

- By adding _(...) around our string we prepare our application for translation process. wxWidgets can help developing multilanguage applications and when some some different language will be used, it will automatically search for translaction of "Label changed" text in strings database.
- wxWidgets may be provided in two versions: with unicode support and without it (ansi build). In case of unicode version, we would have to use unicode strings: **L"Label changed"** and in case of ansi version we would have to use standard string notation: **"Label changed"**. Using **_(...)** macro grants that no matter what wxWidgets version is used, it will always produce proper code.

There's alternative version of string macro which works simillarily to _(...) which is written in form _T(...) or wxT(...). It works like _() one but the string is never translated.

[edit] Reading value from wxTextCtrl

Now let's read something that's written by user. We will use wxTextCtrl (with variable TextCtrl1) for this and show the text using standard message box. Let's double click the **Read text** button and change the code to following:

```
void Tutorial_6Frame::OnButton2Click(wxCommandEvent& event)
{
    wxString Text = TextCtrl1->GetValue();
    wxMessageBox(_("User entered text:\n") + Text);
}
```

In the first line we read value from TextCtrl and store it inside variable of type wxString. wxString is wxWidgets implementation of string object and this library uses it as base for string representation.

In the second line we call **wxMessageBox** function which shows standard message box just like it was modal dialog.

Usually items which provide content entered by user have two member functions: GetValue and SetValue. You can use them to read and write content of such item.

[edit] Changing value of wxGauge

Now let's mix reading and writing of item's value on one function. We will use wxGauge for this. We will use it's two member functions: GetValue and SetValue. I've written ealier that those functions usually exist in items where user can enter some value. Well it's not a gold rule and there are exceptions from it like in case of this item :).

Ok, let's add handler for third button:

```
void Tutorial_6Frame::OnButton3Click(wxCommandEvent& event)
{
    int NewValue = Gauge1->GetValue()+10;
    if ( NewValue > 100 ) NewValue = 0;
    Gauge1->SetValue(NewValue);
}
```

In first line we generate new value of progress by reading current one and adding 10 to it. In second line we prevent the value from getting out of range (by default wxGauge has range set to 0..100 - this can be changed in properties). In third line we write new value into gauge.

[edit] Using value from wxSlider to change font size

Now let's do something more advanced. In our resource there's wxSlider and text saying that it changes font size. We will change font of this label. But we will have to update font size in two steps:

- As long as user draggs the slider we should change font size only
- When user finishes dragging we should layout window because size of text changes

wxSlider provides two events that can be used right for this purpose. First is **EVT_COMMAND_SCROLL_THUMB_TRACK** which is fired while dragging the slider. Second is **EVT_COMMAND_SCROLL_THUMB_RELEASE** which is fired when user finishes dragging.

Because we use events which are not default we would have to add them through properties browser. You can switch between editing standard properties and events by clicking on buttons at the top of browser:

If you switch to events, search for required events and choose **Add new handler** from drop-down list. Here's the code for **EVT_COMMAND_SCROLL_THUMB_TRACK** handler:

```
void Tutorial_6Frame::OnSlider1CmdScrollThumbTrack(wxScrollEvent& event)
{
    wxFont Font = StaticText2->GetFont();
    Font.SetPointSize( Slider1->GetValue() );
    StaticText2->SetFont(Font);
}
```

Here we get font used by StaticText control by using **GetFont()** function, change font's size by using

GetPointSize() and write font back by using **SetFont()** function. **GetFont()** and **SetFont()** are available in most items.

Now let's code **EVT_COMMAND_SCROLL_THUMB_RELEASE**:

```
void Tutorial_6Frame::OnSlider1CmdScrollThumbRelease(wxScrollEvent& event)
{
    Layout();
    GetSizer()->SetSizeHints(this);
}
```

In the first line we adopt positions of items to new environment just like in case of changing label in wxStaticText. In the second line we recalculate minimal size of the window making sure that all items will have enough space.

[edit] Changing item's colour

Last thing we will do in this tutorial is to change colour of some item. As in case of fonts we will change label on the left side of **Change** button. Since we will use standard event of this button, we can add event handler by double-clicking on it. And here's the code:

```
void Tutorial_6Frame::OnButton4Click(wxCommandEvent& event)
{
    wxColour OldColour = StaticText3->GetForegroundColour();
    wxColour NewColour = wxGetColourFromUser(this,OldColour);

    if ( NewColour.IsOk() )
    {
        StaticText3->SetForegroundColour(NewColour);
    }
}
```

At the beginning we read current colour to variable **OldColour**. In next lie we call **wxGetColourFromUser** function which opens dialog where we can choose colour.

The **NewColour.IsOk()** call is required because if user cancels colour selection it will return false.

At the end we set new colour by using **SetForegroundColour** function.

Header file to include in the tut....main.h file: **#include <wx/colordlg.h>**

[edit] More informations

We reached end of this tutorial. I showed only few operations on items and there's much more you can do. For more details you can check wxWidgets' documentation available here.

WxSmith tutorial: Creating items with custom paint and mouse handling

From CodeBlocks

Jump to: navigation, search

[edit] Warning: unfinished

Welcome back :) We've already learned nice techniques of working with wxSmith in previous tutorial and now it's time for something even better. Usually when writing some more advanced application, it turns out that none of items provided by wxWidgets fits our purposes, especially when we need to present some data in graphical form. Hopefully as almost all GUI tolkits, wxWidgets is also prepared for this and gives us possibility to create item with custom drawing and mouse handling - this allows us to create any item we like.

[edit] Creating item with custom paint procedure

Before we start creating our customized item, some background knowledge is needed. First thing you should know is how wxWidgets draws items.

Most of items have custom paint procedures provided by operating system or some other toolkit available in system (like gtk+). In case of such items, when OS posts request to repaint item, either such message is forwarded by wxWidgets back into system or even the whole procedure is done without any wxWidgets' intervention. Unfortunately this means that we can not replace paint procedure in case of all items and even if it would work on some platforms, it won't on others. From items available in wxSmith, only wxPanel is guaranteed to work with custom paint procedure.

Now if we overwrite painting on wxPanel, wxWidgets will post two events on each paint request:

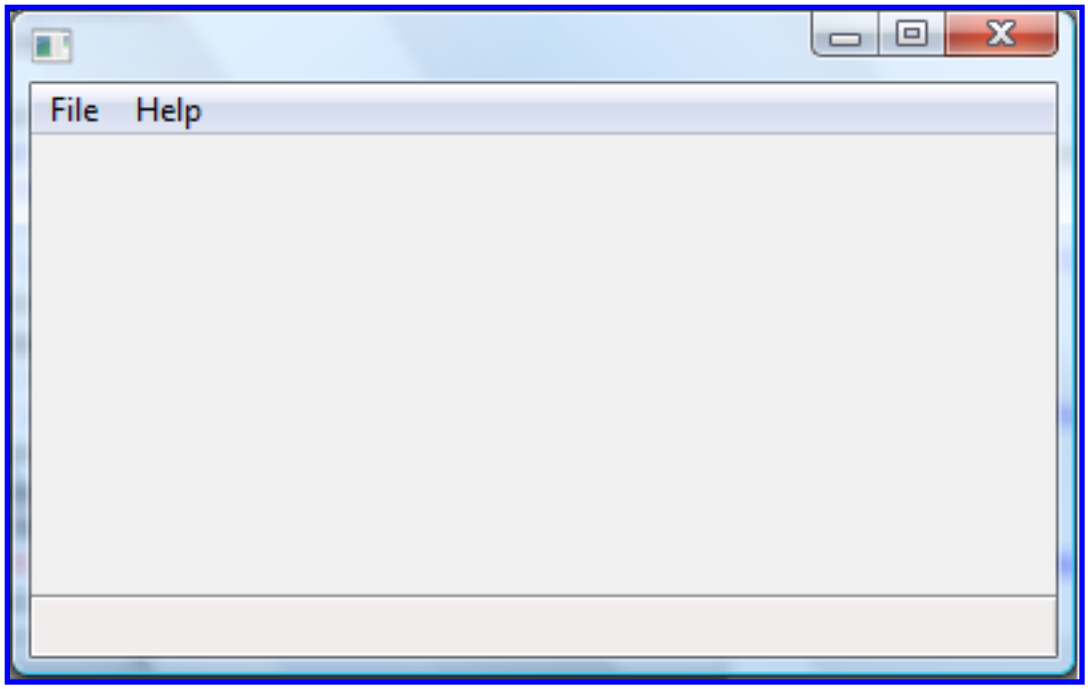
```
* erase background event (EVT_ERASE_BACKGROUND)
* paint event (EVT_PAINT)
```

First one is used to prepare area occupied by our item before adding content and the second one is used to do actual drawing. We don't have to override both events so we can only draw custom background or custom content leaving background as default one.

Second thing you should know before writing custom paint method is that wxWidgets uses wxDC objects to draw. In case of erase background event, this object is provided within event, in case of paint event, we must create such object manually. To get more informations about wxDC and drawing functions, take look at this page in wxWidgets documentation.

[edit] Creating simple resource and overriding paint method

Now let's create some simple window. For our purposes we need some wxPanel inside the window which we will work on:



Now let's go to event's tab in properties browser. You may see that wxPanel can process much more events than other items. That's because it can be use as base for custom items which may have custom handler for any standard event. Ok, let's find EVT_PAINT method (sohuld be at the top) and add new empty event.

The first thing we should do inside the handler is to create wxDC object we will use. If you look at wxWidgets documentation you may read that not doing so causes wxWidgets to go into infinite loop because paint method will be recalled over and over again.

There are few types of wxDC objects we could use here. The most commonly used is wxPaintDC:

```
void Tutorial7Dialog::OnPanel1Paint(wxPaintEvent& event)
```

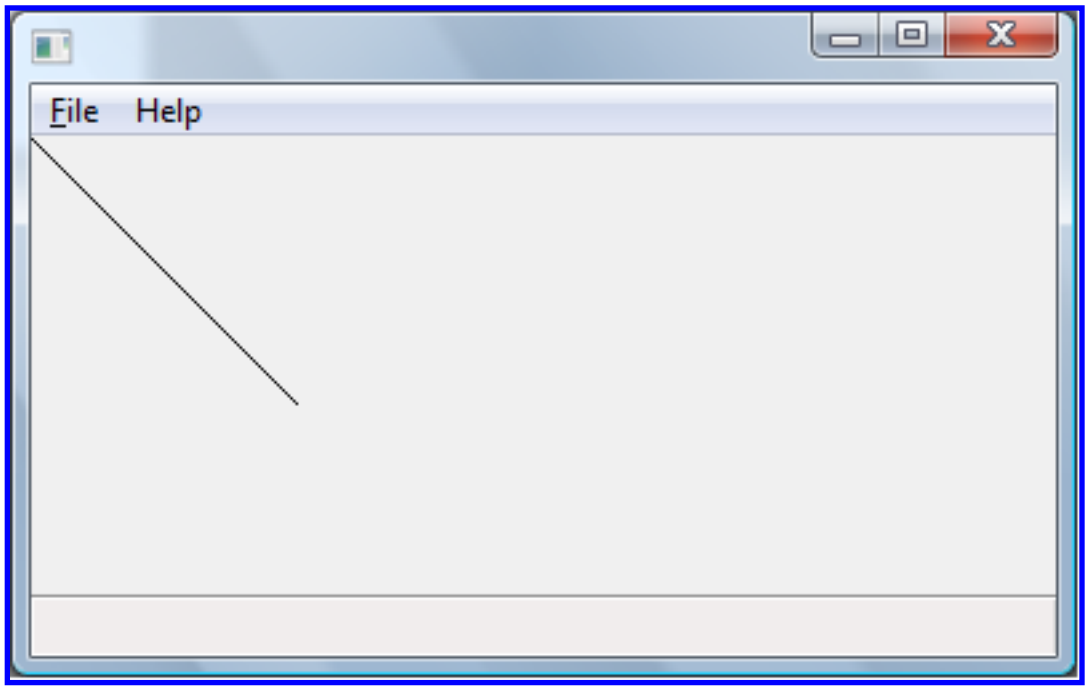
```
{
    wxPaintDC dc( Panel1 );
}
```

You can see that we passed panel's pointer as argument - wxPaintDC require pointer to item we will draw on on it's constructor. Remember not to use *this* opinter.

Now let's add some drawing code - just some diagonal line to test whether drawing does work:

```
void Tutorial7Dialog::OnPanel1Paint(wxPaintEvent& event)
{
    wxPaintDC dc( Panel1 );
    dc.DrawLine( 0, 0, 100, 100 );
}
```

Now if we run our aplication, the window will look like this:



Retrieved from "http://wiki.codeblocks.org/index.php?title=WxSmith_tutorial:_Creating_items_with_custom_paint_and_mouse_handling"

Views

- Article
- Discussion
- Edit

Categories

From CodeBlocks

Jump to: navigation, search

The following categories exist in the wiki.

(first | last) View (previous 50) (next 50) (20 | 50 | 100 | 250 | 500).

- Code::Blocks Contrib Plugins 17 members
- Code::Blocks Core Plugins 1 member
- Code::Blocks Documentation 19 members
- Code::Blocks Plugins 8 members
- Deletion Requests 2 members
- Developer Documentation 24 members
- Installing Code::Blocks 19 members
- Installing Code::Blocks from source 8 members
- Installing Code::Blocks nightly build 6 members
- Installing the latest official version of Code::Blocks 1 member
- Miscellaneous 1 member
- Outdated 5 members
- Plugin development 6 members
- Rewrite needed 1 member
- Roadmaps 3 members
- Scripting Code::Blocks 9 members
- User Documentation 34 members
- User Talk 1 member
- WxSmith Documentation 9 members
- WxSmith extensions 7 members
- 安装Code::Blocks 3 members
- 安装官方最新版本的Code::Blocks 1 member
- 由源代码开始安装Code::Blocks 1 member

(first | last) View (previous 50) (next 50) (20 | 50 | 100 | 250 | 500).
Retrieved from "http://wiki.codeblocks.org/index.php?title=Special:Categories"

Category:WxSmith Documentation

From CodeBlocks

Jump to: navigation, search

Articles related to the wxSmith plugin.

Articles in category "WxSmith Documentation"

There are 9 articles in this category.

A

- Adding new widget to wxSmith

C

- Comparison of wxSmith features

W

- WxSmith tutorial: Accessing items in resource

W cont.

- WxSmith tutorial: Building more complex window
- WxSmith tutorial: Hello world
- WxSmith tutorial: Using wxPanel resources
- WxSmith tutorial: Working with multiple resources

W cont.

- WxSmith tutorials
- Wxsmith tutorial: Working with items

Retrieved from "http://wiki.codeblocks.org/index.php?title=Category:WxSmith_Documentation"

Category: User Documentation

Views

- Category
- Discussion

Login required to edit

From CodeBlocks

Jump to: navigation, search

You have to log in to edit pages.

Return to Talk:WxSmith tutorials.

Retrieved from "http://wiki.codeblocks.org/index.php?title=Talk:WxSmith_tutorials"

Views

- Article
- Discussion
- Edit
- +

Personal tools

- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Search

Go Search

Login required to edit

From CodeBlocks

Jump to: navigation, search

You have to log in to edit pages.

Return to WxSmith tutorials.

Retrieved from "http://wiki.codeblocks.org/index.php?title=WxSmith_tutorials"

Views

- Article
- Discussion
- Edit
- History

Personal tools

- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Search

Go Search

WxSmith tutorials

From CodeBlocks

Revision history
View logs for this page
Jump to: navigation, search
(Latest | Earliest) View (previous 50) (next 50) (20 | 50 | 100 | 250 | 500).

Diff selection: mark the radio boxes of the versions to compare and hit enter or the button at the bottom.
Legend: (cur) = difference with current version, (last) = difference with preceding version, M = minor edit.

Compare selected versions

- (cur) (last) ◉ 18:37, 27 June 2008 Byo (Talk | contribs) (1,106 bytes) (→wxSmith and wxWidgets basics)
- (cur) (last) ◉ ○ 15:30, 19 January 2008 Byo (Talk | contribs) (1,018 bytes) (→wxSmith and wxWidgets basics)
- (cur) (last) ○ ○ 15:14, 19 January 2008 Byo (Talk | contribs) (934 bytes) (→wxSmith and wxWidgets basics)
- (cur) (last) ○ ○ 20:44, 18 January 2008 Byo (Talk | contribs) (842 bytes) (→wxSmith and wxWidgets basics)
- (cur) (last) ○ ○ 12:58, 5 January 2008 Byo (Talk | contribs) (842 bytes)
- (cur) (last) ○ ○ 14:36, 2 January 2008 Byo (Talk | contribs) (748 bytes)
- (cur) (last) ○ ○ 14:35, 2 January 2008 Byo (Talk | contribs) (748 bytes)
- (cur) (last) ○ ○ 14:29, 2 January 2008 Byo (Talk | contribs) (674 bytes)
- (cur) (last) ○ ○ 14:29, 2 January 2008 Byo (Talk | contribs) (642 bytes)
- (cur) (last) ○ ○ 22:26, 5 June 2007 Byo (Talk | contribs)
- (cur) (last) ○ ○ 20:36, 5 June 2007 Byo (Talk | contribs)

Compare selected versions

(Latest | Earliest) View (previous 50) (next 50) (20 | 50 | 100 | 250 | 500).
Retrieved from "http://wiki.codeblocks.org/index.php?title=WxSmith_tutorials"

Views

- Article
- Discussion
- Edit
- History

Personal tools

- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Search

Go Search

Toolbox

- What links here
- Related changes
- RSS Atom
- Upload file
- Special pages

Powered By MediaWiki

- Privacy policy
- About CodeBlocks
- Disclaimers

Log in / create account

From CodeBlocks

Jump to: navigation, search

Log in

You must have cookies enabled to log in to CodeBlocks.

Username:

Password:

☐ Remember my login on this computer

Log in

Retrieved from "http://wiki.codeblocks.org/index.php?title=Special:Userlogin"

Views

- Special

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

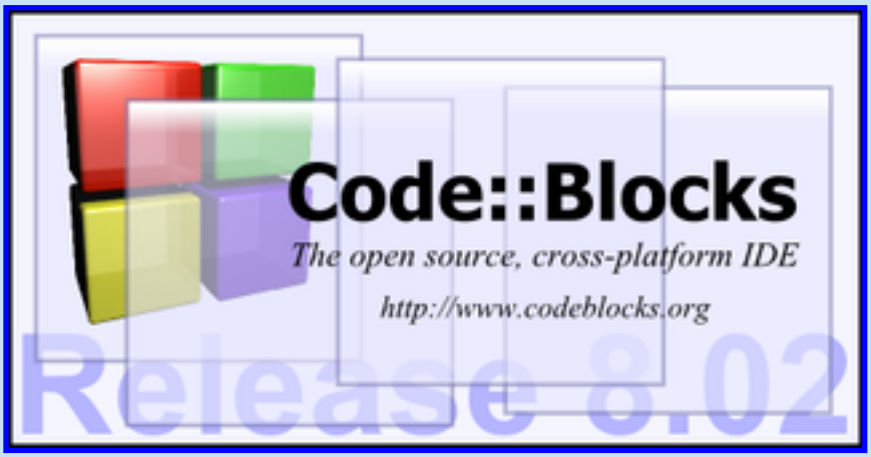
Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help

Main Page

From CodeBlocks

Jump to: navigation, search



Welcome to the official Wiki for Code::Blocks
Code::Blocks is an open-source, cross-platform IDE. Using a plugin architecture, its capabilities and features are defined by the provided plugins.
Currently, Code::Blocks is oriented towards C/C++. The Code::Blocks team does not take responsibility for the content nor accuracy of these pages.

Wiki Editors: In order to login to edit pages, you must create an account on the forums. Use the same username and password for the wiki. Read the Help for editing guidelines. Look in the community portal for things to do.

How do I...

...install Code::Blocks?
Windows · Ubuntu · Mac OS X · Fedora Core · more...

...set up a compiler in Code::Blocks?
MinGW · MS Visual C++ · more...

Download Code::Blocks

- Latest official release: 8.02
- Nightly builds (updated each night)
- Official and third-party binaries (for various distributions)

...create a new project? (Coming soon)

...debug my program? (Coming soon)

...use a 3rd-party library?
wxWidgets · Boost · SDL · more...

Table of Contents

Main article: Code::Blocks Documentation

User documentation

Articles for Code::Blocks users

FAQ

Frequently Asked Questions

Feature List

An index of Code::Blocks' useful features

Code::Blocks Plugins

Plugins extend Code::Blocks' functionality

Off-site documentation

Links to external documentation

Developer documentation

Articles for developers of Code::Blocks itself

Other resources

- Main website
- Community forums

Development

Roadmap

- Developer documentation
- Project page at BerliOS
- **Bugs**: Browse or submit new
- **Features**: Browse or submit new
- **Patches**: Browse or submit new

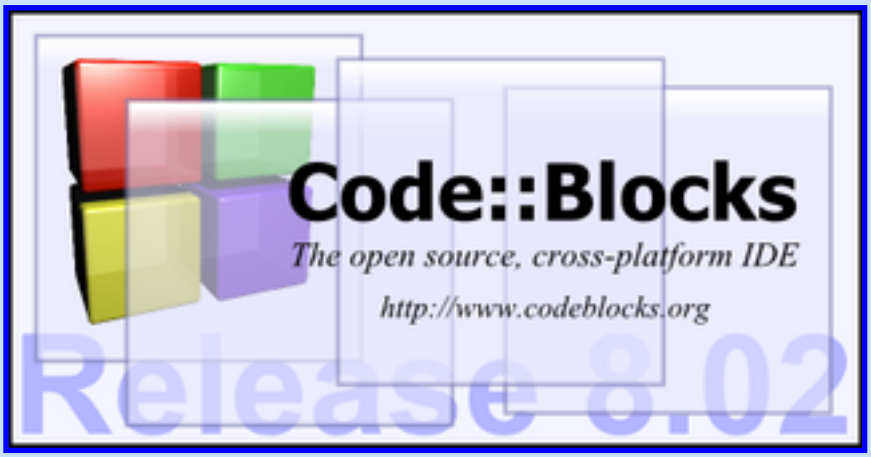
Supported compilers

- GNU GCC (incl. G77) (Linux)
- MinGW GCC (incl. G77) (Win32)
- MSP430 GCC (Win32, Linux, BSD)
- TriCore GCC (Win32, Linux)
- PowerPC GCC (Win32, Linux)
- Apple GCC (Xcode) (Mac OS X)
- Microsoft Visual C++ Toolkit 2003 (Win32)
- Microsoft Visual C++ 2005 (Win32)
- Borland's C++ Compiler 5.5

CodeBlocks:Community Portal

From CodeBlocks

Jump to: navigation, search



Welcome to the Community portal of the Code::Blocks Wiki!

Here you can find Tasks that need to be done and articles that should be written or need improvement. Please help to make this Wiki better! Place requests in the appropriate section. Read the Handbook for editing guidelines. Look at the Help page to learn how to edit the wiki.

This Wiki wants *you*!

Article wishlist

Add your wishes here.

- Running Code::Blocks under VMWare Player. See VMWare Player and freely available virtual OSes for VMWare Player.
- Batch building
- User-defined tools. Little explanation what they are, what the macros mean etc. Add tutorial to the same article about creating a simple tool, for example opening the Windows Explorer / Nautilus to the project directory. Redirects from Tools and External tools.

Check out the forums

News

The Wiki news...

- **11 April:** Started categorizing

- Creating new plugin tutorial. This tutorial should contain information how to build a fully functional plugin with configuration dialog etc. Information how to build and test it on different operating systems.

Edit

Improvable articles

Too short? Spelling, grammar or textual mistakes? Out-of-date? Add these articles here or improve one of them.

Wikify

- Startup script
- Wizard scripts
- Debugger scripts
- Build scripts

- Articles that contain outdated information
- Articles that need a rewrite

Edit

To-Do List

Other things that are necessary...

Main page redesign

The main page needs to be redesigned (hopefully) before the RC3 release. The left-side menu could be tweaked too while we're at it.

Header

the Wiki.

Edit

"Special" pages

- New pages
- Statistics
- Popular pages
- Random page
- All "special" pages

- Remove the blue box
- Replace the header image with a bigger one
- Simplify description

Developers

- Remove Announcements for plugins/patches link

Supported compilers

- Remove it; replace with features (which would include the supported compilers).

Left-side menu

- Remove Random page link
- Add links to basic documentation like features and installing

---

- Add "most wanted" pages
- Categorize those!

Edit

Help with the Wiki

- Code::Blocks Wiki Handbook - Read these guidelines before creating new articles.
- Editing - Learn how to edit the wiki.
- Wikimedia Help - Further information on wiki markup.
- Sandbox - A sandbox to play around in. Make any changes you like here.

Edit

Retrieved from "http://wiki.codeblocks.org/index.php?title=CodeBlocks:Community_Portal"

Views

- Project page
- Discussion
- Edit
- History

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version

Current events

From CodeBlocks

Jump to: navigation, search

[edit] Wiki reorganization / updating for RC3 release

I've started a full reorganization of the Wiki. Almost all pages are divided in to two main categories: User documentation and Developer documentation. I've also added a number of new categories. Some of the older articles I've put in the deletion requests category. Check the editing history for request reasons.

The most biggest change is in the installer documentation. All platforms have 1 documentation for each procedure: installing the latest official version, installing nightly build and installing Code::Blocks from source. Of cource there are documentation lacking in some of the operating systems.

I think it's very important that the Wiki will be updated reflect the changes in RC3. This requires rewriting some parts in almost every article. Luckily the parts that require rewriting are short and the base information of the article can be kept intact.

[edit] Some general tips

- Do **not** start new articles before searching the older ones. Most importantly:
  - Installing Windows nightly build
  - Building Code::Blocks from source (Windows)
  - Building Code::Blocks from source (Linux-General)
- Do **not** announce patches or new plugins in the Wiki. Use the forums instead. Note that information *about* plugins should be in the Wiki.
- Do **not** write articles that contain information about features specific to RC2. RC2 is old and will be replaced soon. Get ready for the next release and write about RC3 instead.

[edit] What can I do?

- Wikify articles; search articles that use HTML code and replace those with Wiki syntax (note that some articles use HTML for "special effects" purposes, do not edit them). Add links to other articles. Make sure that articles use headings, paragraphs and code tags where needed.
- Read articles, follow them and fix any mistakes you find.
- Write about new features in RC3. You can get some ideas for articles from the roadmap. Again,

please search before writing.

When replying to this message, put a "::" before your answer. This will indent the text

like this

which will result in better viewing experience.

Good night and good luck.

--Artojon 17:42, 8 July 2006 (EDT)

Retrieved from "http://wiki.codeblocks.org/index.php?title=Current_events"

Recent changes

From CodeBlocks

Jump to: navigation, search

Track the most recent changes to the wiki on this page.

Below are the last **50** changes in the last last **7** days, as of 20:33, 16 April 2009.
Show last 50 | 100 | 250 | 500 changes in last 1 | 3 | 7 | 14 | 30 days
Hide minor edits | Show bots | Hide anonymous users | Hide logged-in users | Hide patrolled edits | Hide my edits
Show new changes starting from 20:33, 16 April 2009

Namespace: all Go ☐ Invert selection
Retrieved from "http://wiki.codeblocks.org/index.php?title=Special:Recentchanges"

Views

- Special

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Script plugins

From CodeBlocks

Jump to: navigation, search

Script plugins are plugins that are written entirely in script. They are great for smaller-scale plugins or for quick prototyping. Their main practical difference compared to simple scripts is that a script plugin *can add entries in context menus*.

Let's look at a sample plugin (by coincidence, this is shipped with Code::Blocks):

```
// Script plugins must extend cbScriptPlugin

class TestPlugin extends cbScriptPlugin
{
    // mandatory to setup the plugin's info
    constructor()
    {
        info.name = _T("TestPlugin");
        info.title = _T("Test script");
        info.version = _T("0.1a");
        info.license = _T("GPL");
    }

    // optional to create menubar items
    function GetMenu()
    {
        local entries = ::wxArrayString();
        entries.Add(_T("Project/7:-Export Makefile"), 1);
        return entries;
    }

    // optional to create context menu entries
    function GetModuleMenu(who, data)
    {
        local entries = ::wxArrayString();

        if (who == ::mtEditorManager)
        {
            local f = wxFileName();
            f.Assign(data.GetFolder(), ::wxPATH_NATIVE);
            entries.Add(_T("Work with ") + f.GetFullName(), 1);

            entries.Add(_T("Sample entry"), 1);
```

```
        }

        return entries;
    }

    // optional to support ExecutePlugin(pluginNameString)
    function Execute()
    {
        ::ShowMessage(_T("Ho-ho was here ;)"));
        return 0;
    }

    // optional calback for menubar items clicking
    function OnMenuClicked(index)
    {
        if (index == 0)
            ::ShowMessage(_T("Exporting Makefile..."));
    }

    // optional calback for context menu items clicking
    function OnModuleMenuClicked(index)
    {
        if (index == 0)
            ::ShowMessage(_T("Working with file"));
        else if (index == 1)
            ::ShowMessage(_T("Sample entry not working yet"));
        else
            ::ShowMessage(_T("?!? Functionality not implemented yet"));
    }
}

// this call actually registers the script plugin with Code::Blocks
RegisterPlugin(TestPlugin());

// if you want to call this plugin's Execute() function, use this in a script:
// ExecutePlugin(_T("TestPlugin"));
```

As you can see it is well documented so it should be easy to grasp. The one thing that might strike you odd is the following line in `GetMenu()`:

```
entries.Add(_T("Project/7:-Export Makefile"), 1);
```

What "7:-Export Makefile" does is explained in ScriptingManager's notes. In simple words it means "*insert* `'Export Makefile'` menu item at position *7* of the `'Project'` menu, prepending a separator line before the menu item".

Retrieved from "http://wiki.codeblocks.org/index.php?title=Script_plugins"

Help:Contents

From CodeBlocks

Jump to: navigation, search

- Code::Blocks Wiki Handbook - Read these guidelines before creating new articles.
- Editing - Learn how to edit the wiki.
- Wikimedia Help - Further information on wiki markup.
- Sandbox - A sandbox to play around in. Make any changes you like here.

Edit

[edit] IRC Channel

The Code::Blocks IRC channel is #codeblocks [irc.freenode.net]

Retrieved from "http://wiki.codeblocks.org/index.php?title=Help:Contents"

Views

- Help page
- Discussion
- Edit
- History

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal

CodeBlocks:Site support

From CodeBlocks

Jump to: navigation, search

Code::Blocks donation page: http://sourceforge.net/donate/index.php?group_id=126998

Retrieved from "http://wiki.codeblocks.org/index.php?title=CodeBlocks:Site_support"

Views

- Project page
- Discussion
- Edit
- History

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Search

Go Search

WxSmith tutorials

From CodeBlocks

(List of links)
Jump to: navigation, search
< WxSmith tutorials

What links here Namespace: all Go

The following pages link to **WxSmith tutorials**:

View (previous 50) (next 50) (20 | 50 | 100 | 250 | 500).

- WxSmith Tutorial & Pointers
- WxSmith plugin
- User documentation

View (previous 50) (next 50) (20 | 50 | 100 | 250 | 500).
Retrieved from "http://wiki.codeblocks.org/index.php?title=Special:Whatlinkshere"

Views

- Special

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes

Related changes

From CodeBlocks

(to pages linked from "WxSmith tutorials")
Jump to: navigation, search

This special page lists the last changes on pages who are linked. Pages on your watchlist are **bold**.

< WxSmith tutorials

Below are the last **50** changes in the last last **7** days, as of 20:34, 16 April 2009.
Show last 50 | 100 | 250 | 500 changes in last 1 | 3 | 7 | 14 | 30 days
Hide minor edits

No changes on linked pages during the given period.

Retrieved from "http://wiki.codeblocks.org/index.php?title=Special:Recentchangeslinked"

Views

- Special

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Not logged in

From CodeBlocks

Jump to: navigation, search

You must be logged in to upload files.

Return to Main Page.

Views

- Special

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Search

Go Search

Toolbox

Special pages

From CodeBlocks

Jump to: navigation, search

Special pages for all users

- All pages
- Articles with the fewest revisions
- Articles with the most categories
- Articles with the most revisions
- Book sources
- Broken redirects
- Categories
- Dead-end pages
- Disambiguation pages
- Double redirects
- Export pages
- File list
- Gallery of new files
- List of blocked IP addresses and usernames
- List redirects
- Log in / create account
- Logs
- Long pages
- MIME search
- Most linked to categories
- Most linked to images
- Most linked to pages
- My watchlist
- New pages
- Oldest pages
- Orphaned pages
- Pages without language links
- Popular pages
- Preferences

- Prefix index
- Protected pages
- Random page
- Random redirect
- Recent changes
- Search
- Short pages
- Statistics
- System messages
- Uncategorized categories
- Uncategorized images
- Uncategorized pages
- Unused categories
- Unused files
- Unused templates
- Upload file
- User contributions
- User list
- Version
- Wanted categories
- Wanted pages

Retrieved from "http://wiki.codeblocks.org/index.php?title=Special:Specialpages"

Views

- Special

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events

WxSmith tutorials

From CodeBlocks

Welcome to wxSmith tutorials page. Here you can learn how to create GUI applications for wxWidgets platform using wxSmith. At the beginning we will start with some basic tutorials which will show what can you do with wxSmith and how you can create simple applications. I hope that there will also be some advanced tutorials for those who know wxWidgets very well.

wxSmith and wxWidgets basics

On this tutorials you will learn basic things about wxSmith and wxWidgets.

- Tutorial 1: Hello world
- Tutorial 2: Working with items
- Tutorial 3: Building more complex window
- Tutorial 4: Working with multiple resources
- Tutorial 5: Using wxPanel resources
- Tutorial 6: Accessing items in resource
- Tutodial 7: wxSmith tutorial: Creating items with custom paint and mouse handling

Category: WxSmith Documentation

WxSmith tutorials

From CodeBlocks

Revision as of 18:37, 27 June 2008 by Byo (Talk | contribs)
(diff) ←Older revision | Current revision (diff) | Newer revision→ (diff)
Jump to: navigation, search

Welcome to wxSmith tutorials page. Here you can learn how to create GUI applications for wxWidgets platform using wxSmith. At the beginning we will start with some basic tutorials which will show what can you do with wxSmith and how you can create simple applications. I hope that there will also be some advanced tutorials for those who know wxWidgets very well.

[edit] wxSmith and wxWidgets basics

On this tutorials you will learn basic things about wxSmith and wxWidgets.

- Tutorial 1: Hello world
- Tutorial 2: Working with items
- Tutorial 3: Building more complex window
- Tutorial 4: Working with multiple resources
- Tutorial 5: Using wxPanel resources
- Tutorial 6: Accessing items in resource
- Tutodial 7: wxSmith tutorial: Creating items with custom paint and mouse handling

Retrieved from "http://wiki.codeblocks.org/index.php?title=WxSmith_tutorials"

Category: WxSmith Documentation

Views

- Article
- Discussion
- Edit
- History

Personal tools

CodeBlocks:Privacy policy

From CodeBlocks

Jump to: navigation, search
There is currently no text in this page, you can search for this page title in other pages or edit this page.
Retrieved from "http://wiki.codeblocks.org/index.php?title=CodeBlocks:Privacy_policy"

Views

- Project page
- Discussion
- Edit

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Search

Go Search

Toolbox

- What links here

CodeBlocks:About

From CodeBlocks

Jump to: navigation, search

Code::Blocks is a full-featured IDE (Integrated Development Environment) aiming to make the individual developer (and the development team) work in a *nice programming environment* offering everything he/they would ever need from a program of that kind. Its *pluggable* architecture allows you, the developer, to add any kind of functionality to the core program, through the use of plugins...

Views

- Project page
- Discussion
- Edit
- History

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Search

CodeBlocks:General disclaimer

From CodeBlocks

Jump to: navigation, search

The Code::Blocks team does not take responsibility for the content nor accuracy of these pages.

Views

- Project page
- Discussion
- Edit
- History

Personal tools

- 72.137.39.112
- Talk for this IP
- Log in / create account

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

Search

Go Search